Deep Learning CT3 Mini Project

Context:

Assume you are a Data Scientist tasked with solving challenges in the agricultural industry, particularly focusing on the detection of diseases in crops. You have been given the responsibility to build a model that can classify images of beans into various categories to assist farmers in identifying disease-infected crops. You will apply transfer learning using TensorFlow and MobileNet to achieve this goal.

Objective:

The goal of this exam is to assess your ability to apply transfer learning to a real-world agricultural problem. You will load a dataset of bean images, preprocess them, apply a pre-trained MobileNet model, and deploy your solution in a user-friendly application to assist farmers.

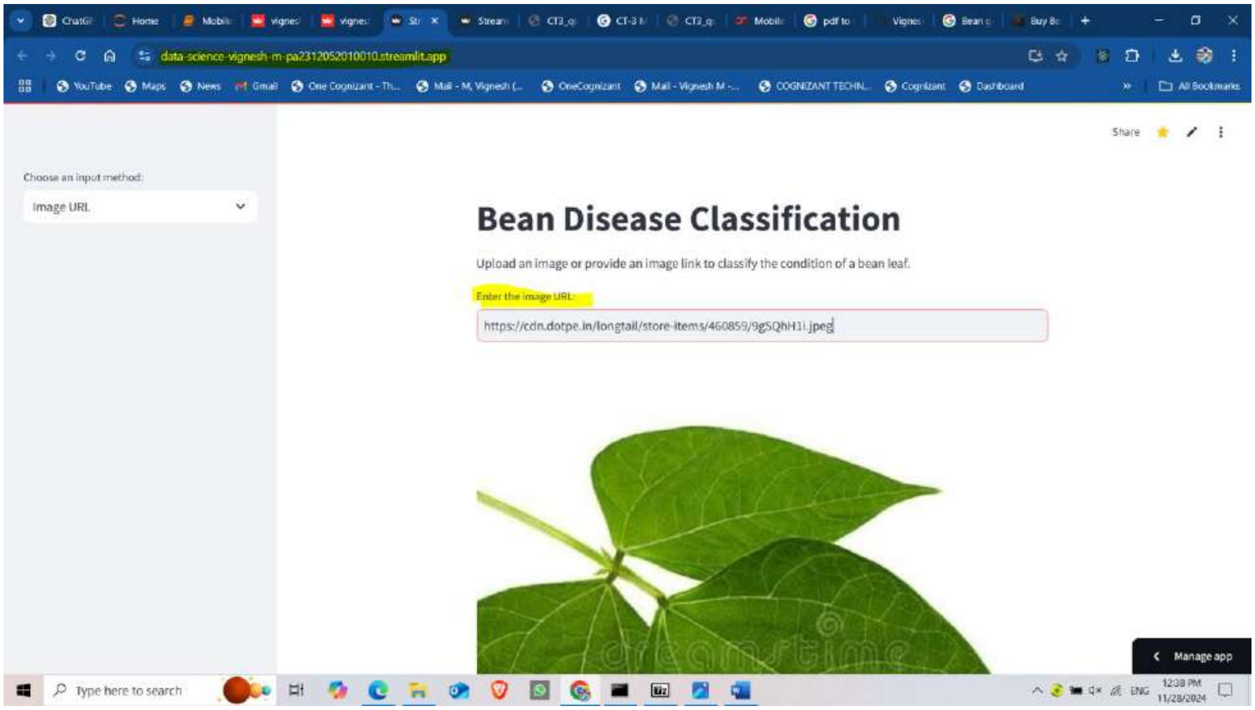
Conclusion:

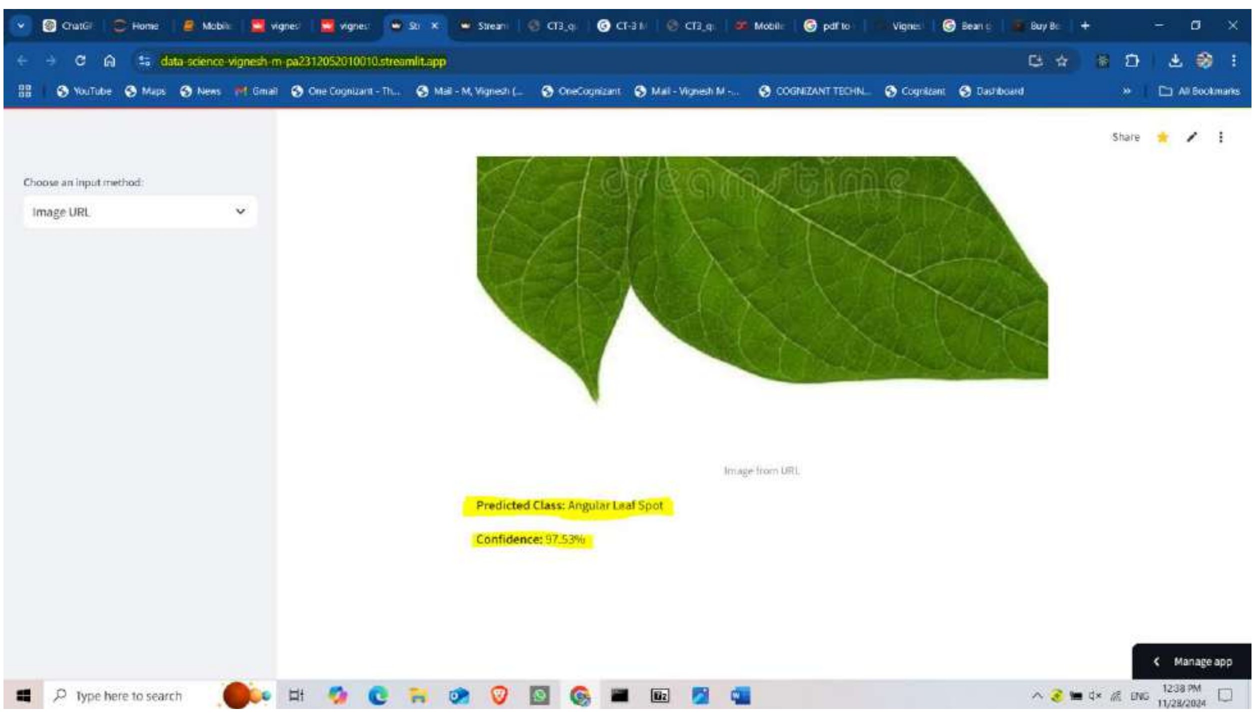
Using transfer learning with the pre-trained MobileNet model, we successfully developed a robust system for classifying bean images to assist farmers in identifying bean types or detecting health issues. By fine-tuning the model with a custom classification head and deploying it in a user-friendly Streamlit application, we bridged the gap between advanced AI techniques and practical agricultural needs. This solution empowers farmers with a fast, accurate, and accessible tool to enhance crop management and decision-making.

The model is deployed the Streamlit app using **Streamlit Community Cloud**

Streamlit link for bean leaf detection:

https://data-science-vignesh-m-pa2312052010010.streamlit.app/





Below code part shows how the dataset set is implemented and devolped MobilNet model for Bean leaf detection

```
import tensorflow as tf
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping
import seaborn as sns
```

```
# Step 1: Load the data
(ds_train, ds_test), ds_info = tfds.load(
    'beans',
    split=['train', 'test'],
    as_supervised=True,
    with_info=True
)
```

```
Downloading and preparing dataset Unknown size (download: Unknown size, generated: 171.63 MiB, total: 171.63 MiB) to /root/tensorflow_da
Dl Completed...: 100%    3/3 [00:11<00:00,  4.50s/ url]

Dl Size...: 100%    170/170 [00:11<00:00, 15.86 MiB/s]
```

```
ds_info
```

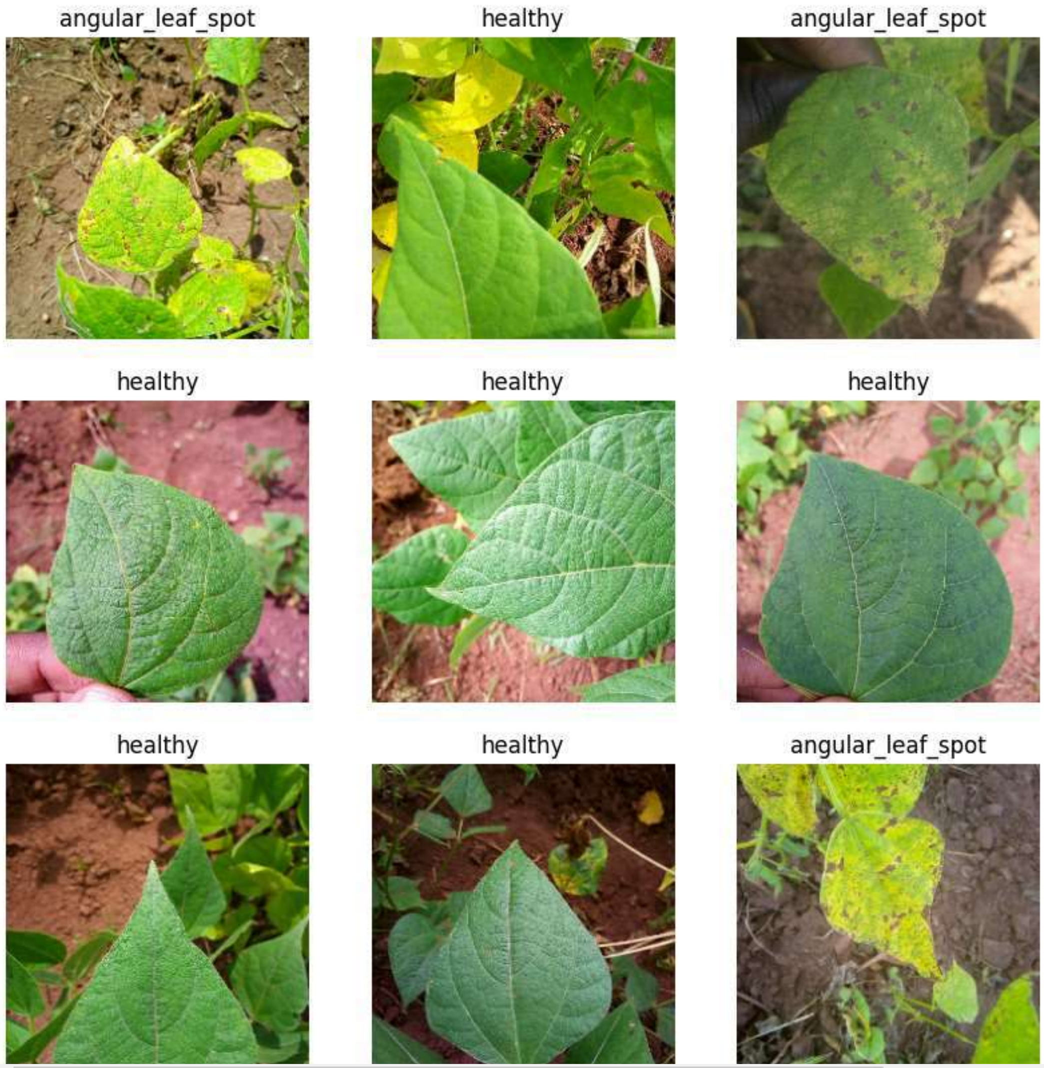
```
tfds.core.DatasetInfo(
    name='beans',
    full_name='beans/0.1.0',
    description="""
    Beans is a dataset of images of beans taken in the field using smartphone
    cameras. It consists of 3 classes: 2 disease classes and the healthy class.
    Diseases depicted include Angular Leaf Spot and Bean Rust. Data was annotated by
    experts from the National Crops Resources Research Institute (NaCRRI) in Uganda
    and collected by the Makerere AI research lab.
    """,
    homepage='https://github.com/AI-Lab-Makerere/ibean/',
    data_dir=PosixGPath('/tmp/tmpack38qbztfds'),
    file_format=tfrecord,
    download_size=Unknown size,
    dataset_size=171.63 MiB,
    features=FeaturesDict({
        'image': Image(shape=(500, 500, 3), dtype=uint8),
        'label': ClassLabel(shape=(), dtype=int64, num_classes=3),
    }),
    supervised_keys=('image', 'label'),
    disable_shuffling=False,
    splits={
        'test': <SplitInfo num_examples=128, num_shards=1>,
        'train': <SplitInfo num_examples=1034, num_shards=2>,
        'validation': <SplitInfo num_examples=133, num_shards=1>,
    },
    citation="""@ONLINE {beansdata,
        author="Makerere AI Lab",
        title="Bean disease dataset",
        month="January",
        year="2020",
        url="https://github.com/AI-Lab-Makerere/ibean/"
    }""",
)
```

```
# Step 2: Preprocess the data
IMG_SIZE = 224  # MobileNet standard input size
BATCH_SIZE = 32

def preprocess(image, label):
    image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))
    image = tf.cast(image, tf.float32) / 255.0  # Normalize to [0, 1] range
    return image, label

ds_train = ds_train.map(preprocess).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
ds_test = ds_test.map(preprocess).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
```

```
# Step 3: Visualize the data
class_names = ds_info.features['label'].names
plt.figure(figsize=(10, 10))
```

```
# Unbatch the dataset for visualization and shuffle it
ds_train_unbatched = ds_train.unbatch().shuffle(buffer_size=1000)

plt.figure(figsize=(10, 10))
for i, (image, label) in enumerate(ds_train_unbatched.take(9)):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(image.numpy())
    plt.title(class_names[label.numpy()])
    plt.axis("off")
plt.show()
```

```
<Figure size 1000x1000 with 0 Axes>
```

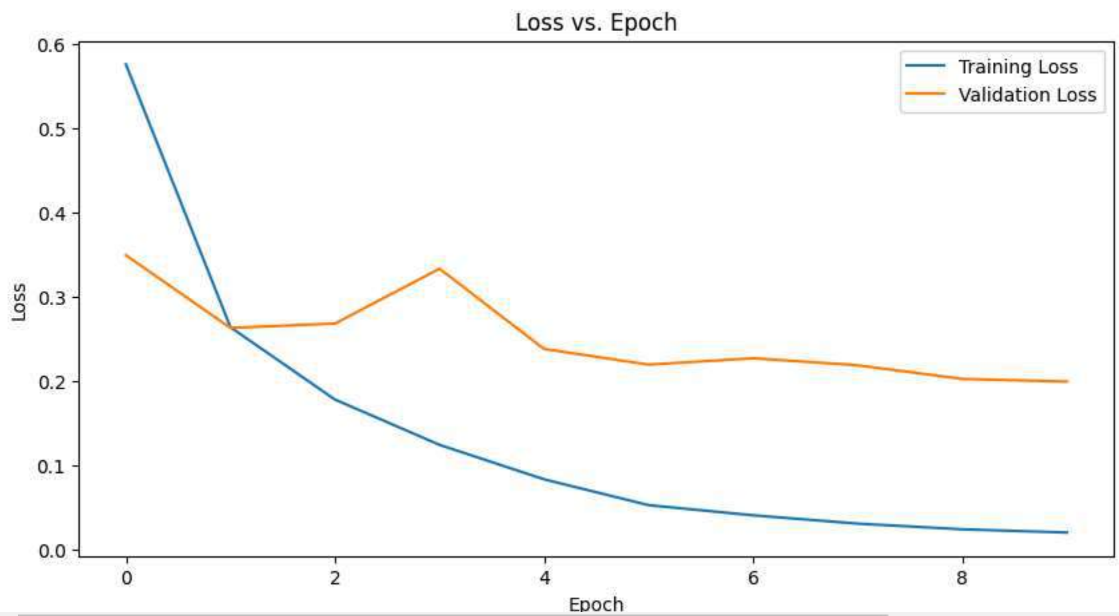



```
# Step 4: Load MobileNet for Transfer Learning
base_model = MobileNetV2(input_shape=(IMG_SIZE, IMG_SIZE, 3), include_top=False, weights='imagenet')
base_model.trainable = False  # Freeze the base model

# Add new classification layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(len(class_names), activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Step 5: Apply Early Stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
```

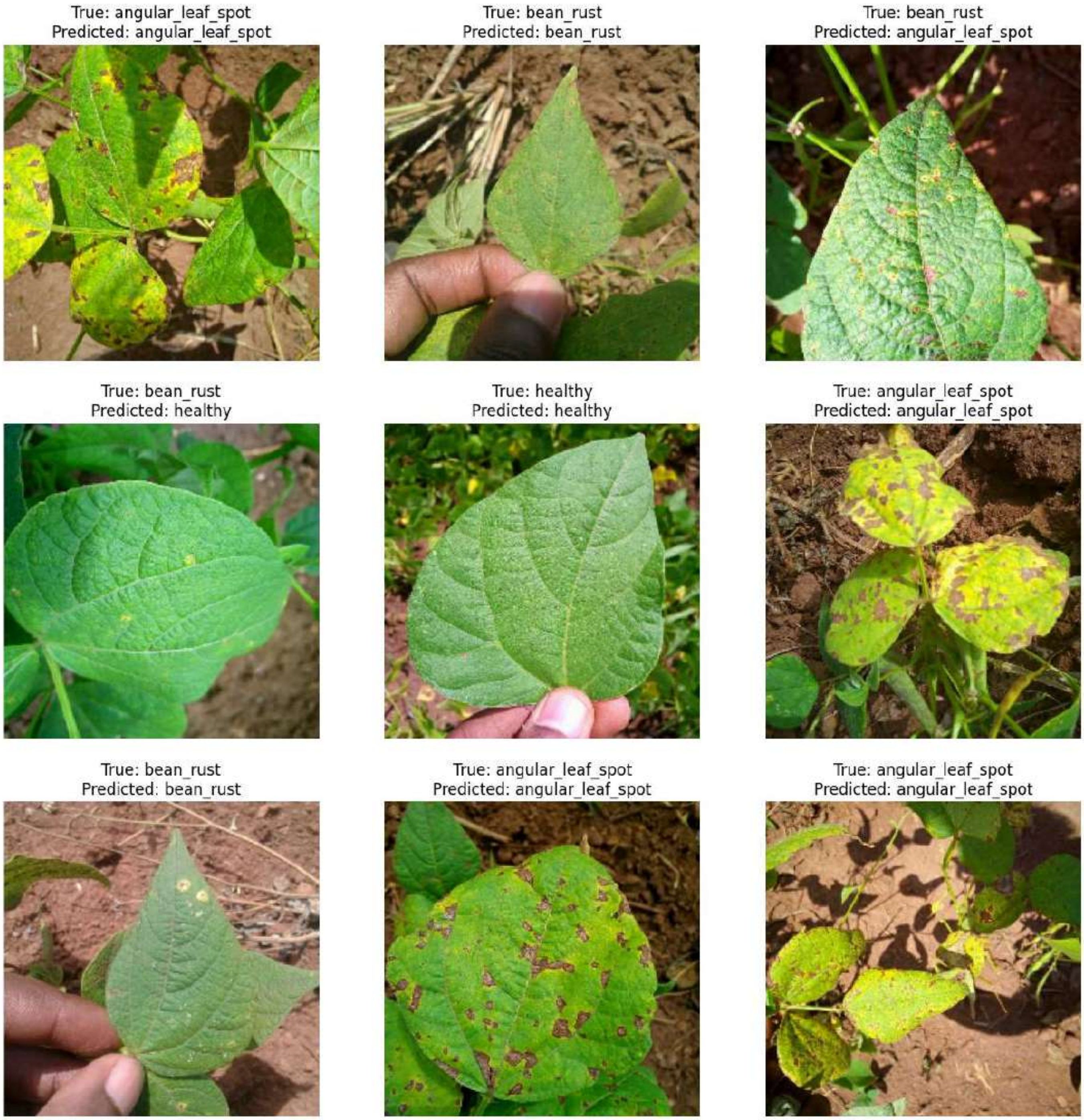
```
# Step 6: Train the Model
history = model.fit(ds_train, epochs=10, validation_data=ds_test, callbacks=[early_stopping])

# Step 7: Plot Loss vs. Epoch
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss vs. Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 18s 249ms/step - accuracy: 0.6835 - loss: 0.7197 - val_accuracy: 0.8594 - val_loss: 0.3496
Epoch 2/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 36ms/step - accuracy: 0.9060 - loss: 0.2514 - val_accuracy: 0.9062 - val_loss: 0.2632
Epoch 3/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9630 - loss: 0.1574 - val_accuracy: 0.8984 - val_loss: 0.2685
Epoch 4/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9641 - loss: 0.1185 - val_accuracy: 0.8672 - val_loss: 0.3335
Epoch 5/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9764 - loss: 0.0869 - val_accuracy: 0.9062 - val_loss: 0.2382
Epoch 6/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step - accuracy: 0.9919 - loss: 0.0511 - val_accuracy: 0.8906 - val_loss: 0.2196
Epoch 7/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 0.9959 - loss: 0.0379 - val_accuracy: 0.9062 - val_loss: 0.2273
Epoch 8/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9966 - loss: 0.0294 - val_accuracy: 0.8828 - val_loss: 0.2188
Epoch 9/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 1.0000 - loss: 0.0228 - val_accuracy: 0.8906 - val_loss: 0.2027
Epoch 10/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 1.0000 - loss: 0.0189 - val_accuracy: 0.9141 - val_loss: 0.1995
```




```
# Step 8: Predict on the Test Data
test_images, test_labels = next(iter(ds_test))
predictions = model.predict(test_images)

# Step 9: Visualize the Results on Test Samples
plt.figure(figsize=(15, 15))
for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(test_images[i].numpy())
    predicted_label = class_names[tf.argmax(predictions[i])]
    true_label = class_names[test_labels[i].numpy()]
    plt.title(f'True: {true_label}\nPredicted: {predicted_label}')
    plt.axis("off")
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
```




```
#Inbuilt model evaluation
# model.evaluate(test_images
# Evaluate the model on the test dataset
test_loss, test_accuracy = model.evaluate(ds_test)

print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

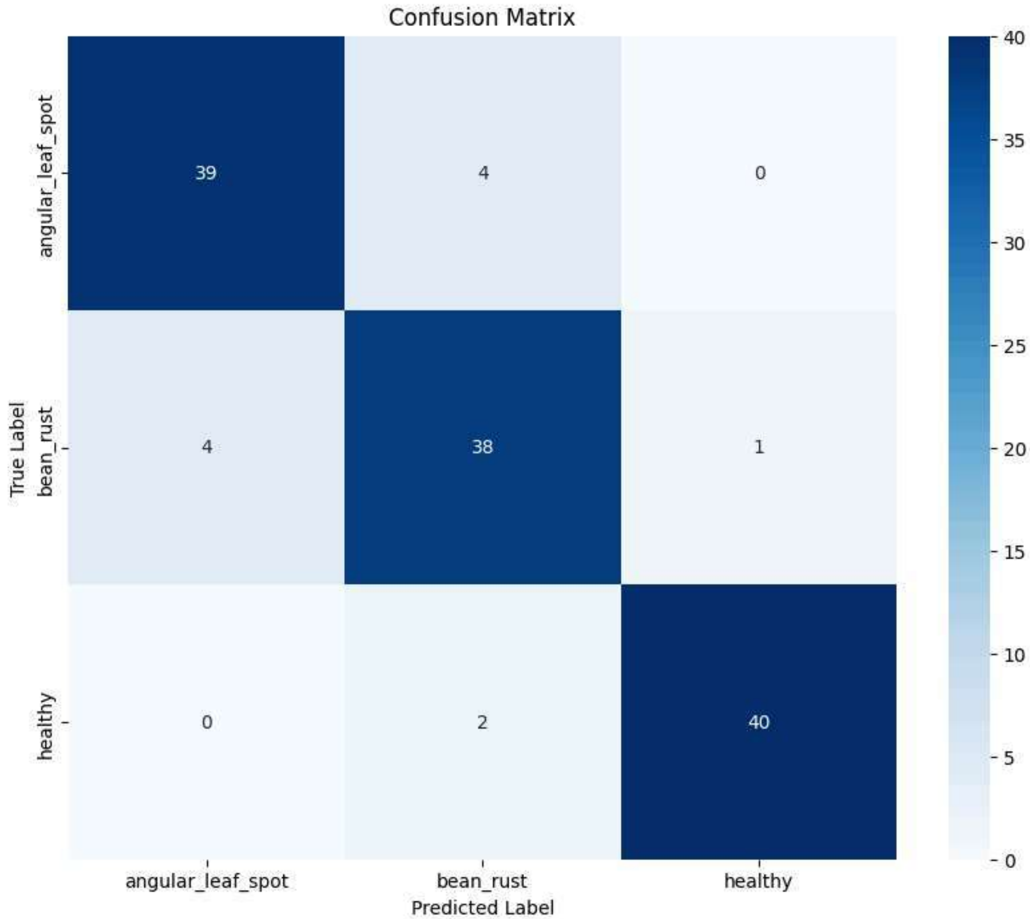
```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.9115 - loss: 0.2059
Test Loss: 0.19950199127197266
Test Accuracy: 0.9140625
```

```
#Evaluate with Custom Metrics
from sklearn.metrics import precision_recall_fscore_support


# Get predictions on the entire test dataset
predictions = model.predict(ds_test)

# Convert predictions to class labels (assuming softmax output)
predicted_classes = tf.argmax(predictions, axis=1).numpy()

# Extract true labels from the test dataset
true_labels = tf.concat([y for x, y in ds_test], axis=0).numpy()

# Compute precision, recall, F1-score for each class
precision, recall, f1, _ = precision_recall_fscore_support(true_labels, predicted_classes)

for i, class_name in enumerate(class_names):
    print(f"Class: {class_name}")
    print(f"  Precision: {precision[i]:.2f}")
    print(f"  Recall:    {recall[i]:.2f}")
    print(f"  F1 Score:  {f1[i]:.2f}")
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step
Class: angular_leaf_spot
  Precision: 0.91
  Recall:    0.91
  F1 Score:  0.91
Class: bean_rust
  Precision: 0.86
  Recall:    0.88
  F1 Score:  0.87
Class: healthy
  Precision: 0.98
  Recall:    0.95
  F1 Score:  0.96
```

```
#Compute Metrics
#Use libraries like scikit-learn for more metrics:
from sklearn.metrics import classification_report, confusion_matrix

# Classification Report
print(classification_report(true_labels, predicted_classes, target_names=class_names))

# Confusion Matrix
cm = confusion_matrix(true_labels, predicted_classes)
print(cm)

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

```
                  precision    recall  f1-score   support

angular_leaf_spot       0.91      0.91      0.91        43
        bean_rust       0.86      0.88      0.87        43
          healthy       0.98      0.95      0.96        42

         accuracy                           0.91       128
        macro avg       0.92      0.91      0.91       128
     weighted avg       0.91      0.91      0.91       128

[[39  4  0]
 [ 4 38  1]
 [ 0  2 40]]
```




```
model.save('C:/Users/Vignesh/Downloads/SRM/Sem/Deep Learning/model/mobilenetv2_beans_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
```

```
model.save('C:/Users/Vignesh/Downloads/SRM/Sem/Deep Learning/model/mobilenetv2_beans_model.keras')
```

```
#Streamlit code for deploy
#streamlit run app.py

import streamlit as st
from tensorflow.keras.models import load_model
from PIL import Image
import numpy as np

# Load the saved model
#MODEL_PATH = 'bean_disease_model.keras'
MODEL_PATH='C:/Users/Vignesh/Downloads/SRM/Sem/Deep Learning/model/mobilenetv2_beans_model.keras'
model = load_model(MODEL_PATH)

# Define the class names (replace with your actual class names)
class_names = ['Healthy', 'Angular Leaf Spot', 'Bean Rust']

# Function to preprocess the uploaded image
def preprocess_image(image):
```

```
    img = image.resize((128, 128))  # Resize to model's expected input size
    img_array = np.array(img) / 255.0  # Normalize pixel values
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    return img_array

# Streamlit App
st.title("Bean Disease Classification")
st.write("Upload an image of a bean leaf to identify its health condition.")

# File uploader
uploaded_file = st.file_uploader("Choose a bean leaf image", type=["jpg", "jpeg", "png"])

if uploaded_file:
    try:
        # Display the uploaded image
        image = Image.open(uploaded_file)
        st.image(image, caption="Uploaded Image", use_column_width=True)

        # Preprocess the image
        processed_image = preprocess_image(image)

        # Make a prediction
        predictions = model.predict(processed_image)
        predicted_class = class_names[np.argmax(predictions)]
        confidence = np.max(predictions) * 100

        # Display the result
        st.write(f"**Predicted Class:** {predicted_class}")
        st.write(f"**Confidence:** {confidence:.2f}%")
    except Exception as e:
        st.error(f"An error occurred: {e}")

# Error handling for unsupported file types
else:
    st.write("Please upload a valid image file.")
```

Start coding or generate with AI.