

Complete Code Implementation

The complete Python code for this project is available below. The implementation uses TensorFlow/Keras for model development and training.

#1. Import Libraries

```
import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import MobileNetV2

from tensorflow.keras.layers import Dense, Dropout,
    GlobalAveragePooling2D

from tensorflow.keras.models import Model

from tensorflow.keras.callbacks import EarlyStopping,
    ReduceLROnPlateau

from sklearn.metrics import classification_report, confusion_matrix

import numpy as np, matplotlib.pyplot as plt, seaborn as sns, os

print("TensorFlow version:", tf.__version__)
```

#2. Correct dataset paths

```
BASE_DIR = "/content/tulsi_leaf_dataset/classifier
model/dataset/train_aug"

OUTPUT_DIR = "/content/tulsi_leaf_dataset/output"
os.makedirs(OUTPUT_DIR, exist_ok=True)
```

```
BATCH_SIZE = 32
```

```
IMG_SIZE = (224, 224)
```

#3. Data generators

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
```

```
        zoom_range=0.2,  
        width_shift_range=0.1,  
        height_shift_range=0.1,  
        shear_range=0.1,  
        horizontal_flip=True,  
        validation_split=0.2  
)  
  
train_gen = train_datagen.flow_from_directory(  
    BASE_DIR,  
    target_size=IMG_SIZE,  
    batch_size=BATCH_SIZE,  
    subset='training',  
    class_mode='categorical'  
)  
val_gen = train_datagen.flow_from_directory(  
    BASE_DIR,  
    target_size=IMG_SIZE,  
    batch_size=BATCH_SIZE,  
    subset='validation',  
    class_mode='categorical',  
    shuffle=False  
)  
  
class_names = list(train_gen.class_indices.keys())  
num_classes = len(class_names)  
print("Classes:", class_names)  
  
#4. Build Transfer Learning model (MobileNetV2)  
base_model = MobileNetV2(input_shape=(224, 224, 3),  
include_top=False, weights='imagenet')
```

```
base_model.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.3)(x)
preds = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=preds)
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e
4),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()

#5. Train model
callbacks = [
    EarlyStopping(patience=5, restore_best_weights=True,
                  monitor='val_accuracy'),
    ReduceLROnPlateau(patience=3, factor=0.5, monitor='val_loss')
]

history = model.fit(
    train_gen,
    validation_data=val_gen,
    epochs=20,
    callbacks=callbacks
)

#6. Fine-tuning (optional)
```

```
base_model.trainable = True
for layer in base_model.layers[:100]:
    layer.trainable = False

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e
5),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

fine_tune_history = model.fit(
    train_gen,
    validation_data=val_gen,
    epochs=10,
    callbacks=callbacks
)

#7. Plot training curves
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title('Accuracy')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Loss')
plt.legend()
plt.show()
```

```
#8. Evaluate model on validation data

val_gen.reset()

preds = model.predict(val_gen, verbose=1)

y_pred = np.argmax(preds, axis=1)

y_true = val_gen.classes

print("\n Classification Report:")

print(classification_report(y_true, y_pred,
target_names=class_names))

cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(6,5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_names, yticklabels=class_names)

plt.xlabel("Predicted")

plt.ylabel("True")

plt.title("Confusion Matrix — Validation Set")

plt.show()

# 9. Save the trained model

MODEL_PATH = os.path.join(OUTPUT_DIR, "tulsi_leaf_mobilenetv2.h5")

model.save(MODEL_PATH)

print("Model saved at:", MODEL_PATH)
```