GangadharSSingh Assignment 04

Leaf Classification Using CNN on the Flavia Dataset

Assignment(Project) Overview

This Assignment aims to build a Convolutional Neural Network (CNN) model to classify plant leaf images from the **Flavia dataset**. The workflow includes data preprocessing, model building, training, evaluation, and performance analysis, comparision between CNN & LSTM and also comparision **with transfer learning via MobileNet model**

Dataset Description

The **Flavia dataset** contains images of various plant leaves captured on a uniform background. Each image represents a unique species, making it ideal for image classification tasks.

- **Format**: JPEG images
- **Number of Classes**: 32 species
- **Original Image Sizes**: Varying

Data Preprocessing

Steps:

- Resize all images to a fixed size (e.g., `128x128`)
- Convert to grayscale
- Normalize pixel values to `[0, 1]`
- One-hot encode labels
- Split dataset:
    - Training: 70%
    - Validation: 15%
    - Testing: 15%

GangadharSShiva Assignment 4

Assignment Questions

Preprocess the data: You will need to preprocess the Flavia dataset by resizing the images to a fixed size, converting them to grayscale, and splitting them into training, validation, and test sets.

Build a CNN model: You will need to design and implement a CNN model architecture that can effectively classify plant leaves based on their images.

Train the model: You will need to train the CNN model on the preprocessed Flavia dataset using appropriate hyperparameters and regularization techniques.

Evaluate the model: You will need to evaluate the performance of the trained CNN model on the test set of the Flavia dataset using appropriate evaluation metrics such as accuracy, precision, recall, and F1 score.

Analyze the results: You will need to analyze the performance of the model and identify any potential areas for improvement. You can visualize the learned features of the model, plot confusion matrices, and perform other analysis techniques to gain insights into the model's behavior.

Question 1 :Preprocess the data: You will need to preprocess the Flavia dataset by resizing the images to a fixed size, converting them to grayscale, and splitting them into training, validation, and test sets.

```
# load leaves folder from google drive

from google.colab import drive
drive.mount('/content/drive')


drive_leaves_dir = '/content/drive/MyDrive/usd-backup/Colab Notebooks/AAI-511/Lea


!ls "{drive_leaves_dir}"
```

```
dataset_dir = drive_leaves_dir
```

```
1180.jpg  1419.jpg  2042.jpg  2281.jpg  2520.jpg  3087.jpg  3326.jpg  3567.j
1181.jpg  1420.jpg  2043.jpg  2282.jpg  2521.jpg  3088.jpg  3327.jpg  3568.j
1182.jpg  1421.jpg  2044.jpg  2283.jpg  2522.jpg  3089.jpg  3328.jpg  3569.j
1183.jpg  1422.jpg  2045.jpg  2284.jpg  2523.jpg  3090.jpg  3329.jpg  3570.j
1184.jpg  1423.jpg  2046.jpg  2285.jpg  2524.jpg  3091.jpg  3330.jpg  3571.j
1185.jpg  1424.jpg  2047.jpg  2286.jpg  2525.jpg  3092.jpg  3331.jpg  3572.j
1186.jpg  1425.jpg  2048.jpg  2287.jpg  2526.jpg  3093.jpg  3332.jpg  3573.j
1187.jpg  1426.jpg  2049.jpg  2288.jpg  2527.jpg  3094.jpg  3333.jpg  3574.j
1188.jpg  1427.jpg  2050.jpg  2289.jpg  2528.jpg  3095.jpg  3334.jpg  3575.j
1189.jpg  1428.jpg  2051.jpg  2290.jpg  2529.jpg  3096.jpg  3335.jpg  3576.j
1190.jpg  1429.jpg  2052.jpg  2291.jpg  2530.jpg  3097.jpg  3336.jpg  3577.j
1191.jpg  1430.jpg  2053.jpg  2292.jpg  2531.jpg  3098.jpg  3337.jpg  3578.j
1192.jpg  1431.jpg  2054.jpg  2293.jpg  2532.jpg  3099.jpg  3338.jpg  3579.j
1193.jpg  1432.jpg  2055.jpg  2294.jpg  2533.jpg  3100.jpg  3339.jpg  3580.j
1194.jpg  1433.jpg  2056.jpg  2295.jpg  2534.jpg  3101.jpg  3340.jpg  3581.j
1195.jpg  1434.jpg  2057.jpg  2296.jpg  2535.jpg  3102.jpg  3341.jpg  3582.j
1196.jpg  1435.jpg  2058.jpg  2297.jpg  2536.jpg  3103.jpg  3342.jpg  3583.j
1197.jpg  1436.jpg  2059.jpg  2298.jpg  2537.jpg  3104.jpg  3343.jpg  3584.j
1198.jpg  1437.jpg  2060.jpg  2299.jpg  2538.jpg  3105.jpg  3344.jpg  3585.j
1199.jpg  1438.jpg  2061.jpg  2300.jpg  2539.jpg  3106.jpg  3345.jpg  3586.j
1200.jpg  1439.jpg  2062.jpg  2301.jpg  2540.jpg  3107.jpg  3346.jpg  3587.j
1201.jpg  1440.jpg  2063.jpg  2302.jpg  2541.jpg  3108.jpg  3347.jpg  3588.j
1202.jpg  1441.jpg  2064.jpg  2303.jpg  2542.jpg  3109.jpg  3348.jpg  3589.j
1203.jpg  1442.jpg  2065.jpg  2304.jpg  2543.jpg  3110.jpg  3349.jpg  3590.j
1204.jpg  1443.jpg  2066.jpg  2305.jpg  2544.jpg  3111.jpg  3350.jpg  3591.j
1205.jpg  1444.jpg  2067.jpg  2306.jpg  2545.jpg  3112.jpg  3351.jpg  3592.j
1206.jpg  1445.jpg  2068.jpg  2307.jpg  2546.jpg  3113.jpg  3352.jpg  3593.j
1207.jpg  1446.jpg  2069.jpg  2308.jpg  2547.jpg  3114.jpg  3353.jpg  3594.j
1208.jpg  1447.jpg  2070.jpg  2309.jpg  2548.jpg  3115.jpg  3354.jpg  3595.j
1209.jpg  1448.jpg  2071.jpg  2310.jpg  2549.jpg  3116.jpg  3355.jpg  3596.j
1210.jpg  1449.jpg  2072.jpg  2311.jpg  2550.jpg  3117.jpg  3356.jpg  3597.j
1211.jpg  1450.jpg  2073.jpg  2312.jpg  2551.jpg  3118.jpg  3357.jpg  3598.j
1212.jpg  1451.jpg  2074.jpg  2313.jpg  2552.jpg  3119.jpg  3358.jpg  3599.j
1213.jpg  1452.jpg  2075.jpg  2314.jpg  2553.jpg  3120.jpg  3359.jpg  3600.j
1214.jpg  1453.jpg  2076.jpg  2315.jpg  2554.jpg  3121.jpg  3360.jpg  3601.j
1215.jpg  1454.jpg  2077.jpg  2316.jpg  2555.jpg  3122.jpg  3361.jpg  3602.j
1216.jpg  1455.jpg  2078.jpg  2317.jpg  2556.jpg  3123.jpg  3362.jpg  3603.j
1217.jpg  1456.jpg  2079.jpg  2318.jpg  2557.jpg  3124.jpg  3363.jpg  3604.j
1218.jpg  1457.jpg  2080.jpg  2319.jpg  2558.jpg  3125.jpg  3364.jpg  3605.j
1219.jpg  1458.jpg  2081.jpg  2320.jpg  2559.jpg  3126.jpg  3365.jpg  3606.j
1220.jpg  1459.jpg  2082.jpg  2321.jpg  2560.jpg  3127.jpg  3366.jpg  3607.j
1221.jpg  1460.jpg  2083.jpg  2322.jpg  2561.jpg  3128.jpg  3367.jpg  3608.j
1222.jpg  1461.jpg  2084.jpg  2323.jpg  2562.jpg  3129.jpg  3368.jpg  3609.j
1223.jpg  1462.jpg  2085.jpg  2324.jpg  2563.jpg  3130.jpg  3369.jpg  3610.j
1224.jpg  1463.jpg  2086.jpg  2325.jpg  2564.jpg  3131.jpg  3370.jpg  3611.j
1225.jpg  1464.jpg  2087.jpg  2326.jpg  2565.jpg  3132.jpg  3371.jpg  3612.j
```

```
1226.jpg  1465.jpg  2088.jpg  2327.jpg  2566.jpg  3133.jpg  3372.jpg  3613.j
1227.jpg  1466.jpg  2089.jpg  2328.jpg  2567.jpg  3134.jpg  3373.jpg  3614.j
1228.jpg  1467.jpg  2090.jpg  2329.jpg  2568.jpg  3135.jpg  3374.jpg  3615.j
1229.jpg  1468.jpg  2091.jpg  2330.jpg  2569.jpg  3136.jpg  3375.jpg  3616.j
1230.jpg  1469.jpg  2092.jpg  2331.jpg  2570.jpg  3137.jpg  3376.jpg  3617.j
1231.jpg  1470.jpg  2093.jpg  2332.jpg  2571.jpg  3138.jpg  3377.jpg  3618.j
1232.jpg  1471.jpg  2094.jpg  2333.jpg  2572.jpg  3139.jpg  3378.jpg  3619.j
1233.jpg  1472.jpg  2095.jpg  2334.jpg  2573.jpg  3140.jpg  3379.jpg  3620.j
1234.jpg  1473.jpg  2096.jpg  2335.jpg  2574.jpg  3141.jpg  3380.jpg  3621.j
1235.jpg  1474.jpg  2097.jpg  2336.jpg  2575.jpg  3142.jpg  3381.jpg  all.cs
1236.jpg  1475.jpg  2098.jpg  2337.jpg  2576.jpg  3143.jpg  3382.jpg
1237.jpg  1476.jpg  2099.jpg  2338.jpg  2577.jpg  3144.jpg  3383.jpg
1238.jpg  1477.jpg  2100.jpg  2339.jpg  2578.jpg  3145.jpg  3384.jpg
```

```
#/content/drive/MyDrive/usd-backup/Colab Notebooks/AAI-511/Leaves.folder

from IPython.display import Image, display

# Define the path to the image
image_path = '/content/drive/MyDrive/usd-backup/Colab Notebooks/AAI-511/Leaves/10

# Display the image
display(Image(filename=image_path,width=200))
```

```
#

leaf_filenames = ['1001.jpg', '2002.jpg', '3003.jpg', '3001.jpg']
for filename in leaf_filenames:
  image_path = f'{drive_leaves_dir}/{filename}'
  display(Image(filename=image_path,width=200))
```

```
# Y Column is the Class Label, ID Column is the name of the file in the leaves fo

import pandas as pd

# Load the CSV file into a pandas DataFrame
metadata_path = '/content/drive/MyDrive/usd-backup/Colab Notebooks/AAI-511/Leaves
metadata_df = pd.read_csv(metadata_path)

# Display the first few rows of the DataFrame
print(metadata_df.head())
```

```
   Unnamed: 0        id   y
0           0  1300.jpg   5
1           1  3152.jpg  23
2           2  1439.jpg   9
3           3  1243.jpg   4
4           4  1186.jpg   3
```

Question 2 Build a CNN model: You will need to design and implement a CNN model architecture that can effectively classify plant leaves based on their images.

```
# Preprocess the Flavia dataset by resizing the images to a fixed size,
# Converting them to grayscale,
# Splitting them into training, validation, and test sets.


#  Data Preprocessing

import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical
from IPython.display import Image, display #
import pandas as pd #
import matplotlib.pyplot as plt #



LEAF_IMAGE_SIZE = (128, 128)
```

```
# Function takes a image, converts to greyscale, removes rgb, resizes and
# normalizes

def leaf_preprocess_image(image_path, target_size):
    img = cv2.imread(image_path)
    if img is None:
        print(f"Error: Could not read image file: {image_path}") # Added a warnin
        return None
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Convert to grayscale
    img = cv2.resize(img, target_size) # Resize
    img = img / 255.0  # Normalize
    return img

images = []
labels = []
# List all files in the dataset directory that end with .jpg
# ( directory has both jpg and csv file)
image_files = [f for f in os.listdir(dataset_dir) if f.lower().endswith('.jpg')]

print(f"Found {len(image_files)} image files in the directory.")

metadata_df['id_stripped'] = metadata_df['id'].str.strip()
#store the image to labelling mapping
image_to_label = dict(zip(metadata_df['id_stripped'], metadata_df['y']))


# Create a mapping from unique labels to contiguous integer indices
unique_labels = sorted(metadata_df['y'].unique())
species_to_int = {species: i for i, species in enumerate(unique_labels)}
print('Species to Int Mapping{species_to_int }')

num_classes = len(unique_labels)

matched_images_count = 0
for img_file in image_files:
    # Strip whitespace from img_file for robust matching
    img_file_stripped = img_file.strip()
    # Check if the stripped image file exists in the metadata DataFrame
    if img_file_stripped in image_to_label:
        img_path = os.path.join(dataset_dir, img_file)
        processed_img = leaf_preprocess_image(img_path, LEAF_IMAGE_SIZE)

        if processed_img is not None:
            images.append(processed_img)
            species = image_to_label[img_file_stripped] # Use stripped filename f
```

```
            labels.append(species_to_int[species])
            matched_images_count += 1
    else:
        print(f"Warning: No metadata found for image file: {img_file}") # Added a

print(f"Matched {matched_images_count} image files with metadata.")

images = np.array(images)
labels = np.array(labels)

# Add a channel dimension for grayscale images
if images.ndim == 3 and images.shape[0] > 0: # Check if images array is not empty
    images = np.expand_dims(images, axis=-1)
elif images.shape[0] > 0: # If already grayscale but needs channel dim (shape is
     images = np.expand_dims(images, axis=-1)

# One-hot encode labels
if len(labels) > 0: # Check if labels array is not empty
    labels_categorical = to_categorical(labels, num_classes=num_classes)
else:
    labels_categorical = np.array([]) # Initialize as empty array if no labels

# Split data
if images.shape[0] > 0: # Check if there are samples before splitting
    X_train, X_temp, y_train, y_temp = train_test_split(images, labels_categorica
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5
    print(f"Training data shape: {X_train.shape}")
    print(f"Validation data shape: {X_val.shape}")
    print(f"Test data shape: {X_test.shape}")
    print(f"Number of classes: {num_classes}")

    # # Build a CNN model:
    # # design and implement a CNN model architecture that can effectively classi
    # leaves based on their images , display some test image
    # use metadata_df for classes determination

    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(LEAF_IMAGE_SIZE[0], LE
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
```

```
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    model.summary()

    # --- Display a test image (from the test set) ---
    if X_test.shape[0] > 0:
        random_index = random.randint(0, X_test.shape[0] - 1)
        test_image = X_test[random_index]
        true_label_encoded = y_test[random_index]
        true_label_index = np.argmax(true_label_encoded)
        # Get the original species label from the unique_labels list
        true_species = unique_labels[true_label_index]

        # Display the image (need to remove the channel dimension for display)
        display_image = (test_image * 255).astype(np.uint8).squeeze()
        # Use matplotlib to display the image
        plt.imshow(display_image, cmap='gray')
        plt.title(f"Sample Test Image\nTrue species: {true_species}")
        plt.axis('off')
        plt.show()

    else:
        print("No test images available to display.")
else:
    print("No images were loaded and processed successfully. Cannot build and tra
```

```
Found 1907 image files in the directory.
Matched 1907 image files with metadata.
Training data shape: (1334, 128, 128, 1)
Validation data shape: (286, 128, 128, 1)
Test data shape: (287, 128, 128, 1)
Number of classes: 32
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_cc
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 32)	320
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0

conv2d_1 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 128)	3,211,392
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 32)	4,128

Total params: 3,308,192 (12.62 MB)
Trainable params: 3,308,192 (12.62 MB)
Non-trainable params: 0 (0.00 B)

Dataset Summary: Flavia Leaf Classification

Image Summary

- **Total Images Found**: 1,907
- **Images Matched with Metadata**: 1,907
- The dataset is complete and correctly labeled.

Data Split

Dataset	Number of Images	Shape per Image	Description
Training	1,334	(128, 128, 1)	Used for model training (~70% of total data)
Validation	286	(128, 128, 1)	Used for hyperparameter tuning (~15%)
Testing	287	(128, 128, 1)	Used to evaluate model performance (~15%)

- All images are:
  - **Grayscale** (`1` channel)
  - **Resized** to `128x128` pixels during preprocessing

Classification Details

- **Number of Classes**: 32
- Each class corresponds to a **unique plant species**
- This is a **multi-class classification problem**
- Recommended final model layer:

```
Dense(32, activation='softmax')
```

CNN Model Architecture Summary

This is the architecture of the CNN model implemented using Keras' `Sequential` API for classifying 32 species of plant leaves from the Flavia dataset.

Layer-by-Layer Breakdown

Layer Type	Output Shape	Parameters	Description
Conv2D (32 filters)	`(None, 126, 126, 32)`	320	Applies 32 3x3 filters to the input grayscale image (128x128x
MaxPooling2D	`(None, 63, 63, 32)`	0	Downsamples by a factor of 2 (2x2 pool size).
Conv2D (64 filters)	`(None, 61, 61, 64)`	18,496	Applies 64 3x3 filters, further reducing spatial size.
MaxPooling2D	`(None, 30, 30, 64)`	0	Downsamples again by 2x2 pooling.
Conv2D (128 filters)	`(None, 28, 28, 128)`	73,856	Deeper feature extraction with 128 filters.
MaxPooling2D	`(None, 14, 14, 128)`	0	Downsamples to 14x14x128.
Flatten	`(None, 25088)`	0	Converts 3D feature map into a 1D vector for the Dense layer
Dense (128 units)	`(None, 128)`	3,211,392	Fully connected layer with 128 neurons.
Dropout (rate=0.5)	`(None, 128)`	0	Regularization to prevent overfitting.
Dense (32 units)	`(None, 32)`	4,128	Output layer with softmax activation for 32 classes.

Model Summary

- **Total Parameters**: `3,308,192`
- **Trainable Parameters**: `3,308,192`
- **Non-trainable Parameters**: `0`
- **Model Size**: ~12.6 MB

Interpretation

- The model consists of **3 convolutional layers**, each followed by **max pooling**, to extract spatial features.
- A **flatten layer** prepares the data for fully connected layers.
- The **dense layer** with 128 units learns high-level patterns.
- **Dropout** is used for regularization.
- Final **dense layer with 32 units** maps features to one of the 32 plant species using **softmax activation**.

Suitable For:

- Multi-class classification (32 plant species)
- Grayscale images resized to 128x128
- Dataset like Flavia with modest size and high intra-class similarity

#

Below Cell Block answers 3,4,5.

Question 3: Train the model: You will need to train the CNN model on the preprocessed Flavia dataset using appropriate hyperparameters and regularization techniques.

Question 4Evaluate the model: You will need to evaluate the performance of the trained CNN model on the test set of the Flavia dataset using appropriate evaluation metrics such as accuracy, precision, recall, and F1 score.

Question 5 Analyze the results: You will need to analyze the performance of the model and identify any potential areas for improvement. You can visualize the learned features of the model, plot confusion matrices, and perform other analysis techniques to gain insights into the model's behavior.

```
#  train the CNN model on the preprocessed Flavia dataset using appropriate hyper

import matplotlib.pyplot as plt
import numpy as np
import kerastuner as kt
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Sequential # Import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
import tensorflow as tf # Import tensorflow


# Tune and identify the best model


def build_tunable_model(hp):
    model = Sequential()
    model.add(Conv2D(hp.Int('conv_1_filter', min_value=32, max_value=128, step=16
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(hp.Int('conv_2_filter', min_value=64, max_value=256, step=32
    model.add(MaxPooling2D((2, 2)))
    # Add optional third convolutional layer
```

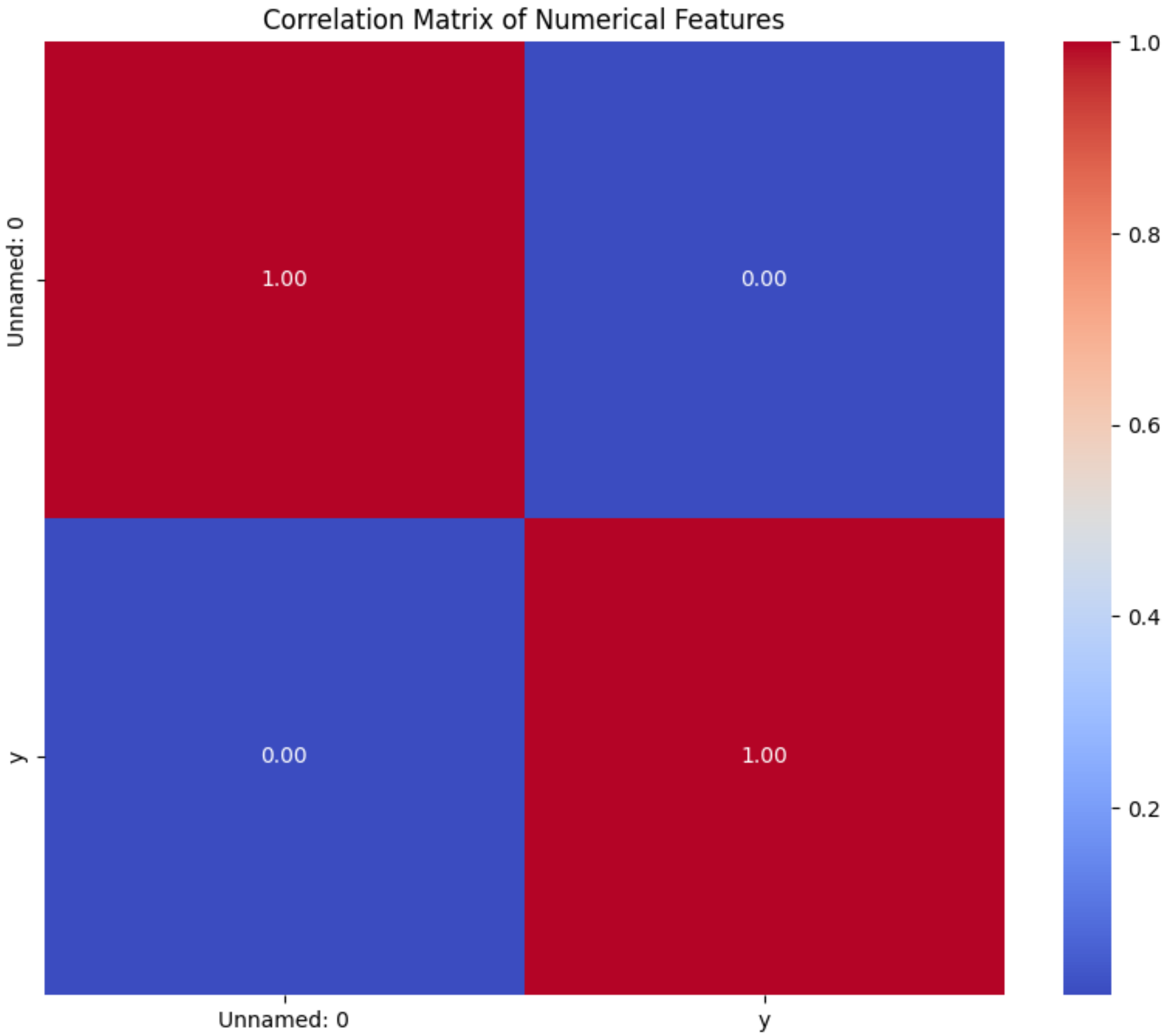
```
    if hp.Boolean("use_conv_3"):
        model.add(Conv2D(hp.Int('conv_3_filter', min_value=128, max_value=512, st
        model.add(MaxPooling2D((2, 2)))

    model.add(Flatten())
    model.add(Dense(hp.Int('dense_1_units', min_value=64, max_value=512, step=32)
    model.add(Dropout(hp.Float('dropout', min_value=0.0, max_value=0.5, step=0.1)
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Set up the tuner
tuner = kt.Hyperband(
    build_tunable_model,
    objective='val_accuracy',
    max_epochs=3,
    factor=3,
    directory='my_dir',
    project_name='intro_to_kt')

# Define early stopping callback
early_stopping = EarlyStopping(
    monitor='val_loss',  # Monitor validation loss
    patience=5,          # Number of epochs with no improvement
    restore_best_weights=True
)


# Run the hyperparameter search
if 'X_train' in locals() and X_train.shape[0] > 0:
    print("Starting hyperparameter tuning...")
    tuner.search(X_train, y_train,
                 epochs=3,
                 validation_data=(X_val, y_val),
                 callbacks=[early_stopping]) # Pass the early stopping callback

    # Get the best hyperparameters
    best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]

    print(f"""
    optimal number of filters in the first conv layer is {best_hps.get('conv_1_fi
    optimal number of filters in the second conv layer is {best_hps.get('conv_2_f
```

```
    Whether to use a third conv layer is {best_hps.get('use_conv_3')}.
    optimal number of filters in the third conv layer is {best_hps.get('conv_3_fi
    optimal number of units in the first dense layer is {best_hps.get('dense_1_un
    optimal dropout rate is {best_hps.get('dropout')}.
    """)

    # Build the best model
    best_model = tuner.get_best_models(num_models=1)[0]

    # Evaluate the best model on the test data
    if 'X_test' in locals() and X_test.shape[0] > 0:
        print("Evaluating the best model on test data...")
        loss, accuracy = best_model.evaluate(X_test, y_test, verbose=0)

        print(f"Test Loss: {loss:.4f}")
        print(f"Test Accuracy: {accuracy:.4f}")
    else:
        print("No test data available to evaluate the model.")
else:
    print("Training data is not loaded correctly, cannot perform hyperparameter t
```

```
Trial 30 Complete [00h 01m 12s]
val_accuracy: 0.8986014127731323

Best val_accuracy So Far: 0.9160839319229126
Total elapsed time: 00h 31m 26s

    The optimal number of filters in the first conv layer is 96.
    The optimal number of filters in the second conv layer is 128.
    Whether to use a third conv layer is True.
    The optimal number of filters in the third conv layer is 128.
    The optimal number of units in the first dense layer is 192.
    The optimal dropout rate is 0.30000000000000004.

Evaluating the best model on test data...
Test Loss: 0.6464
Test Accuracy: 0.8780
```

```
# Plot correlation matix

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

numerical_df = metadata_df.select_dtypes(include=np.number)
```

```
# Calculate the correlation matrix
correlation_matrix = numerical_df.corr()

# Plot the correlation matrix using seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```




```
# Evaluate the performance of the trained CNN model on the test set of the Flavia
# such as accuracy, precision, recall, and F1 score, and plot the graphs

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
# Evaluate the best model on the test data and plot results
```

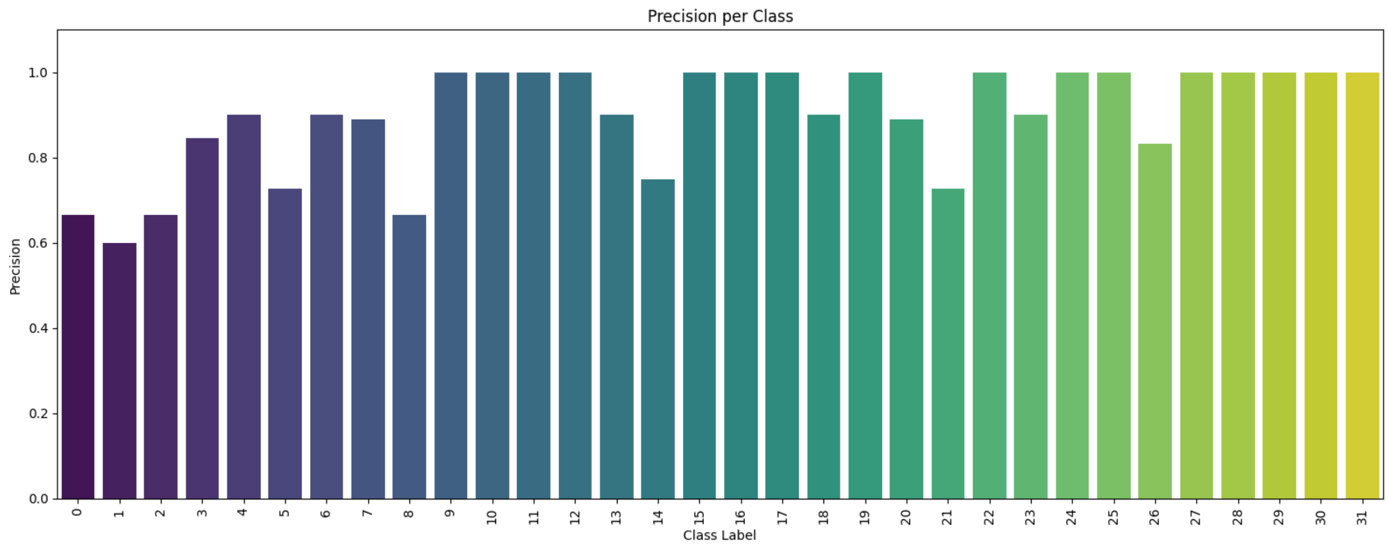
```
if 'best_model' in locals() and 'X_test' in locals() and 'y_test' in locals() and
    print("\n--- Model Evaluation on Test Set ---")

    # Evaluate the model to get loss and accuracy
    loss, accuracy = best_model.evaluate(X_test, y_test, verbose=0)
    print(f"Test Loss: {loss:.4f}")
    print(f"Test Accuracy: {accuracy:.4f}")

    # Make predictions to calculate classification metrics
    y_pred_probs = best_model.predict(X_test)
    y_pred_classes = np.argmax(y_pred_probs, axis=1)
    y_true_classes = np.argmax(y_test, axis=1)

    # Ensure unique_labels is available and corresponds to the integer indices
    if 'unique_labels' in locals():
        target_names = [str(label) for label in unique_labels]
    else:
        # Fallback if unique_labels is not available, use integer class names
        target_names = [str(i) for i in range(num_classes)]

    # Determine which classes are actually present in the true and predicted labe
    present_classes_indices = np.unique(np.concatenate((y_true_classes, y_pred_cl
    filtered_target_names = [target_names[i] for i in present_classes_indices]

    # Generate Classification Report (Precision, Recall, F1-score)
    print("\nClassification Report:")
    report = classification_report(y_true_classes, y_pred_classes,
                                   labels=present_classes_indices,
                                   target_names=filtered_target_names,
                                   output_dict=True, # Get output as dictionary
                                   zero_division=0)

    # Print the report in a readable format
    print(classification_report(y_true_classes, y_pred_classes,
                               labels=present_classes_indices,
                               target_names=filtered_target_names,
                               zero_division=0))



    # Extract metrics for plotting (excluding 'accuracy', 'macro avg', 'weighted 
    metrics_data = {label: report[label] for label in filtered_target_names}

    # Create DataFrames for plotting
```

```
metrics_df = pd.DataFrame(metrics_data).T[['precision', 'recall', 'f1-score']
metrics_df = metrics_df.reset_index().rename(columns={'index': 'Class'})


# Plot Precision
plt.figure(figsize=(15, 6))
sns.barplot(x='Class', y='precision', data=metrics_df, palette='viridis')
plt.title('Precision per Class')
plt.ylabel('Precision')
plt.xlabel('Class Label')
plt.xticks(rotation=90)
plt.ylim(0, 1.1) # Ensure y-axis starts from 0 and goes slightly above 1
plt.tight_layout()
plt.show()

# Plot Recall
plt.figure(figsize=(15, 6))
sns.barplot(x='Class', y='recall', data=metrics_df, palette='magma')
plt.title('Recall per Class')
plt.ylabel('Recall')
plt.xlabel('Class Label')
plt.xticks(rotation=90)
plt.ylim(0, 1.1)
plt.tight_layout()
plt.show()

# Plot F1-score
plt.figure(figsize=(15, 6))
sns.barplot(x='Class', y='f1-score', data=metrics_df, palette='cividis')
plt.title('F1-score per Class')
plt.ylabel('F1-score')
plt.xlabel('Class Label')
plt.xticks(rotation=90)
plt.ylim(0, 1.1)
plt.tight_layout()
plt.show()

# Plot Confusion Matrix
print("\nPlotting Confusion Matrix...")
conf_matrix = confusion_matrix(y_true_classes, y_pred_classes, labels=present_

plt.figure(figsize=(18, 16)) # Increased figure size for better readability
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=filtered_target_names,
            yticklabels=filtered_target_names)
```

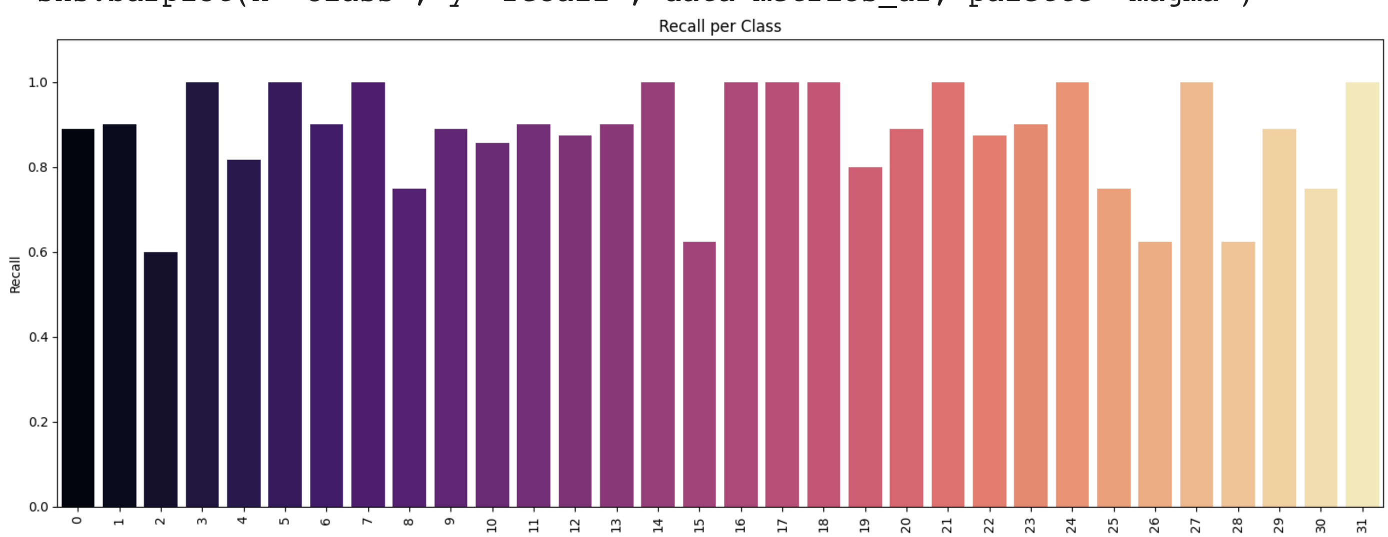
```
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.title('Confusion Matrix')
    plt.tight_layout() # Adjust layout to prevent labels from overlapping
    plt.show()


else:
    print("\nEvaluation Skipped ")
    print("Model, test data, or necessary variables are not available to perform 
```

```
--- Model Evaluation on Test Set ---
Test Loss: 0.6464
Test Accuracy: 0.8780
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step

Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.89      0.76         9
           1       0.60      0.90      0.72        10
           2       0.67      0.60      0.63        10
           3       0.85      1.00      0.92        11
           4       0.90      0.82      0.86        11
           5       0.73      1.00      0.84         8
           6       0.90      0.90      0.90        10
           7       0.89      1.00      0.94         8
           8       0.67      0.75      0.71         8
           9       1.00      0.89      0.94         9
          10       1.00      0.86      0.92         7
          11       1.00      0.90      0.95        10
          12       1.00      0.88      0.93         8
          13       0.90      0.90      0.90        10
          14       0.75      1.00      0.86         9
          15       1.00      0.62      0.77         8
          16       1.00      1.00      1.00        12
          17       1.00      1.00      1.00        10
          18       0.90      1.00      0.95         9
          19       1.00      0.80      0.89        10
          20       0.89      0.89      0.89         9
          21       0.73      1.00      0.84         8
          22       1.00      0.88      0.93         8
          23       0.90      0.90      0.90        10
          24       1.00      1.00      1.00         8
          25       1.00      0.75      0.86         8
          26       0.83      0.62      0.71         8
```

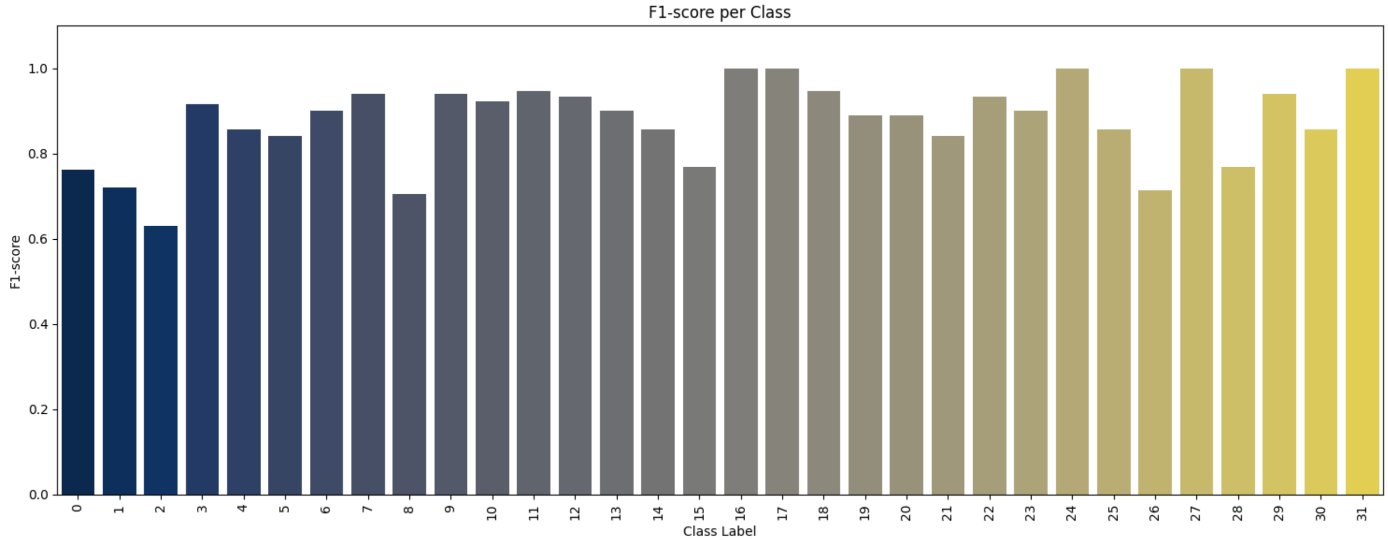
```
          27       1.00      1.00      1.00         8
          28       1.00      0.62      0.77         8
          29       1.00      0.89      0.94         9
          30       1.00      0.75      0.86         8
          31       1.00      1.00      1.00         8

    accuracy                           0.88       287
   macro avg       0.90      0.88      0.88       287
weighted avg       0.90      0.88      0.88       287

/tmp/ipython-input-13-4146123997.py:61: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed ir

  sns.barplot(x='Class', y='precision', data=metrics_df, palette='viridis')
```



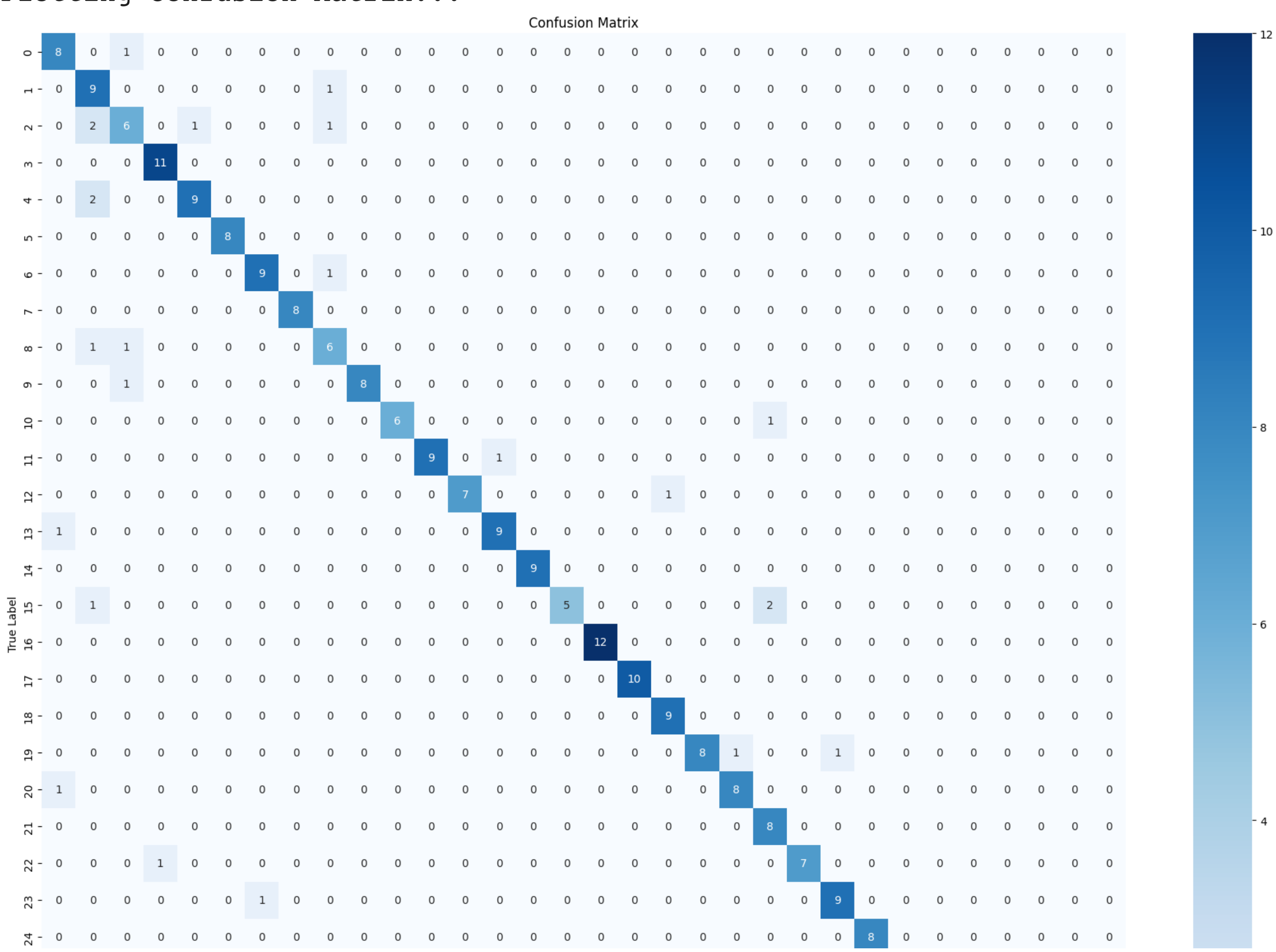

```
/tmp/ipython-input-13-4146123997.py:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed ir

  sns.barplot(x='Class', y='recall', data=metrics_df, palette='magma')
```

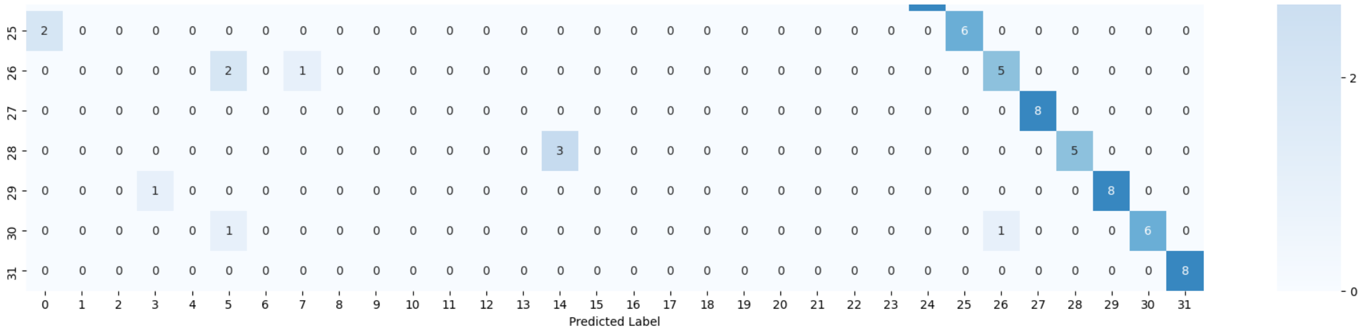


```
/tmp/ipython-input-13-4146123997.py:83: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed ir

  sns.barplot(x='Class', y='f1-score', data=metrics_df, palette='cividis')
```




```
Plotting Confusion Matrix...
```




25 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0
26 0 0 0 0 0 2 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0
27 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0
28 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0
29 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 0 0 8 0 0
30 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 1 0 0 0 6 0
31 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Predicted Label
2
0

Experiment creating an LSM model for Images and Compare with CNN Model

```
# implement the modeling with lstm and compare the accuracy between cnn and lstm

import matplotlib.pyplot as plt
import numpy as np
# --- Implement LSTM Model and Compare Accuracy ---

# Check if data is available from previous steps
if 'X_train' in locals() and X_train.shape[0] > 0:
    print("Data available for LSTM modeling.")

    X_train_lstm = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.sh
    X_val_lstm = X_val.reshape(X_val.shape[0], X_val.shape[1], X_val.shape[2])
    X_test_lstm = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2

    print(f"LSTM Training data shape: {X_train_lstm.shape}")
    print(f"LSTM Validation data shape: {X_val_lstm.shape}")
    print(f"LSTM Test data shape: {X_test_lstm.shape}")


    # Build LSTM Model
    from tensorflow.keras.layers import LSTM

    lstm_model = Sequential([
        LSTM(128, input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])), #
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    lstm_model.compile(optimizer='adam',
```

```
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])

print("\nLSTM Model Summary:")
lstm_model.summary()

# Train the LSTM Model
print("\nTraining LSTM model...")
lstm_history = lstm_model.fit(X_train_lstm, y_train,
                              epochs=50, #
                              batch_size=32, #
                              validation_data=(X_val_lstm, y_val),
                              callbacks=[early_stopping])

# Evaluate the LSTM Model
print("\nEvaluating LSTM model on test data...")
lstm_loss, lstm_accuracy = lstm_model.evaluate(X_test_lstm, y_test, verbose=0

print(f"LSTM Test Loss: {lstm_loss:.4f}")
print(f"LSTM Test Accuracy: {lstm_accuracy:.4f}")

# Compare Accuracies
print("\n--- Accuracy Comparison ---")
# Assuming 'best_model' and 'accuracy' from CNN evaluation are available
if 'accuracy' in locals():
    print(f"CNN Test Accuracy: {accuracy:.4f}")
    print(f"LSTM Test Accuracy: {lstm_accuracy:.4f}")

    # Plotting comparison
    labels = ['CNN', 'LSTM']
    accuracies = [accuracy, lstm_accuracy]

    plt.figure(figsize=(6, 4))
    plt.bar(labels, accuracies, color=['skyblue', 'lightgreen'])
    plt.ylim(0, 1) # Accuracy is between 0 and 1
    plt.ylabel('Test Accuracy')
    plt.title('CNN vs LSTM Test Accuracy')
    for i, acc in enumerate(accuracies):
        plt.text(i, acc + 0.02, f'{acc:.4f}', ha='center')
    plt.show()

    # You can also compare loss
    if 'loss' in locals():
         print("\n--- Loss Comparison ---")
         print(f"CNN Test Loss: {loss:.4f}")
```

```
            print(f"LSTM Test Loss: {lstm_loss:.4f}")

            losses = [loss, lstm_loss]
            plt.figure(figsize=(6, 4))
            plt.bar(labels, losses, color=['skyblue', 'lightgreen'])
            plt.ylabel('Test Loss')
            plt.title('CNN vs LSTM Test Loss')
            for i, lss in enumerate(losses):
                plt.text(i, lss + 0.01, f'{lss:.4f}', ha='center')
            plt.show()


        # Classification Report and Confusion Matrix for LSTM
        print("\n--- LSTM Evaluation ---")
        print("Generating Classification Report and Confusion Matrix for LSTM..."

        y_pred_lstm = lstm_model.predict(X_test_lstm)
        y_pred_classes_lstm = np.argmax(y_pred_lstm, axis=1)
        # y_true_classes is the same for both models

        print("\nLSTM Classification Report:")
        # Ensure unique_labels and filtered_target_names are available
        if 'unique_labels' in locals() and 'filtered_target_names' in locals() an
            print(classification_report(y_true_classes, y_pred_classes_lstm,
                                        labels=present_classes_indices,
                                        target_names=filtered_target_names,
                                        zero_division=0))
            conf_matrix_lstm = confusion_matrix(y_true_classes, y_pred_classes_ls

            plt.figure(figsize=(12, 10))
            sns.heatmap(conf_matrix_lstm, annot=True, fmt='d', cmap='Blues',
                        xticklabels=filtered_target_names,
                        yticklabels=filtered_target_names)
            plt.xlabel('Predicted Label')
            plt.ylabel('True Label')
            plt.title('LSTM Confusion Matrix')
            plt.show()
        else:
             print("Cannot generate detailed report and confusion matrix for LSTM


    else:
        print("CNN accuracy not found. Ensure the CNN evaluation step ran success

else:
```

```
    print("Training data (X_train, y_train) is not available. Cannot build and tr
    print("Please ensure the data preprocessing steps ran correctly.")
```

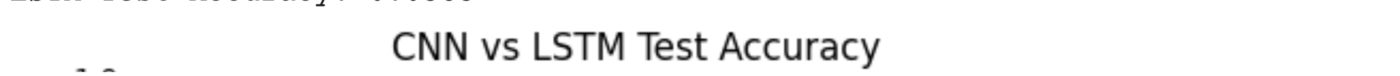
```
Data available for LSTM modeling.
LSTM Training data shape: (1334, 128, 128)
LSTM Validation data shape: (286, 128, 128)
LSTM Test data shape: (287, 128, 128)

LSTM Model Summary:
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserW
  super().__init__(**kwargs)
```

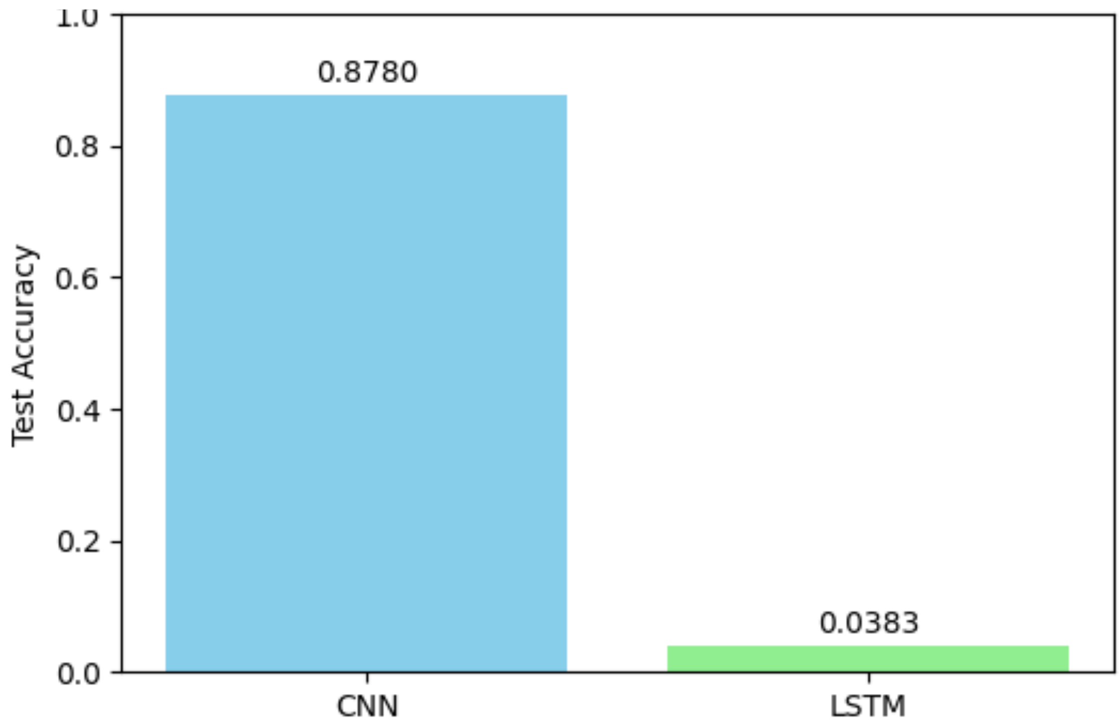
Model: "sequential_1"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 128)	131,584
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 32)	4,128

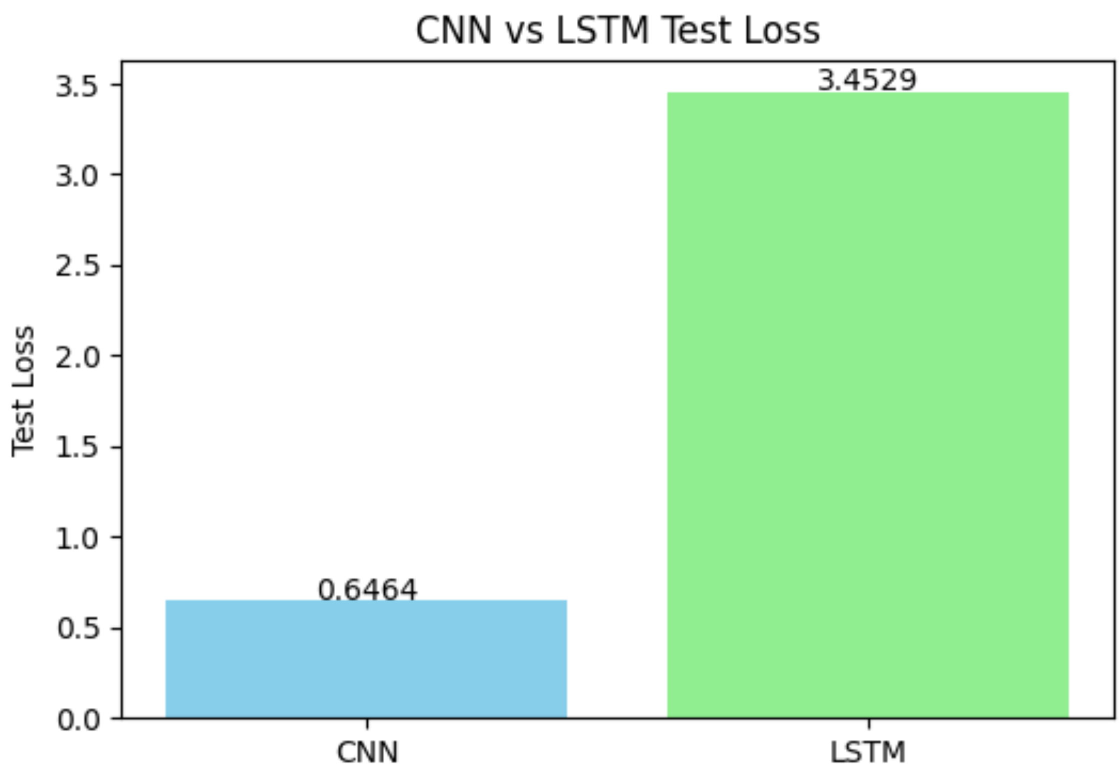
Total params: 135,712 (530.12 KB)
Trainable params: 135,712 (530.12 KB)
Non-trainable params: 0 (0.00 B)

```
Training LSTM model...
Epoch 1/50
42/42 ━━━━━━━━━━━━━━━━━━━━ 5s 27ms/step - accuracy: 0.0303 - loss: 3.5899 - va
Epoch 2/50
42/42 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.0337 - loss: 3.4800 - va
Epoch 3/50
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.0479 - loss: 3.4472 - va
Epoch 4/50
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.0314 - loss: 3.4734 - va
Epoch 5/50
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.0292 - loss: 3.4774 - va
Epoch 6/50
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.0283 - loss: 3.4693 - va
Epoch 7/50
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.0357 - loss: 3.4624 - va

Evaluating LSTM model on test data...
LSTM Test Loss: 3.4529
LSTM Test Accuracy: 0.0383

--- Accuracy Comparison ---
CNN Test Accuracy: 0.8780
LSTM Test Accuracy: 0.0383
```



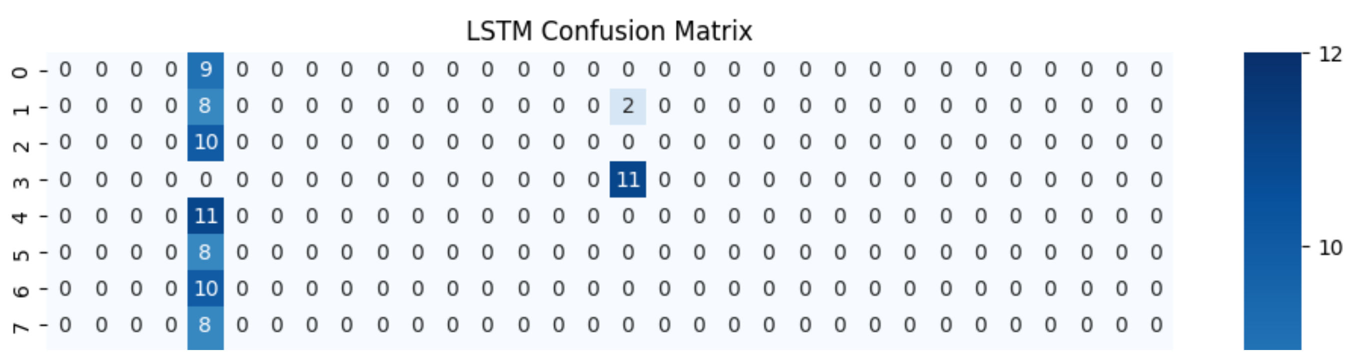


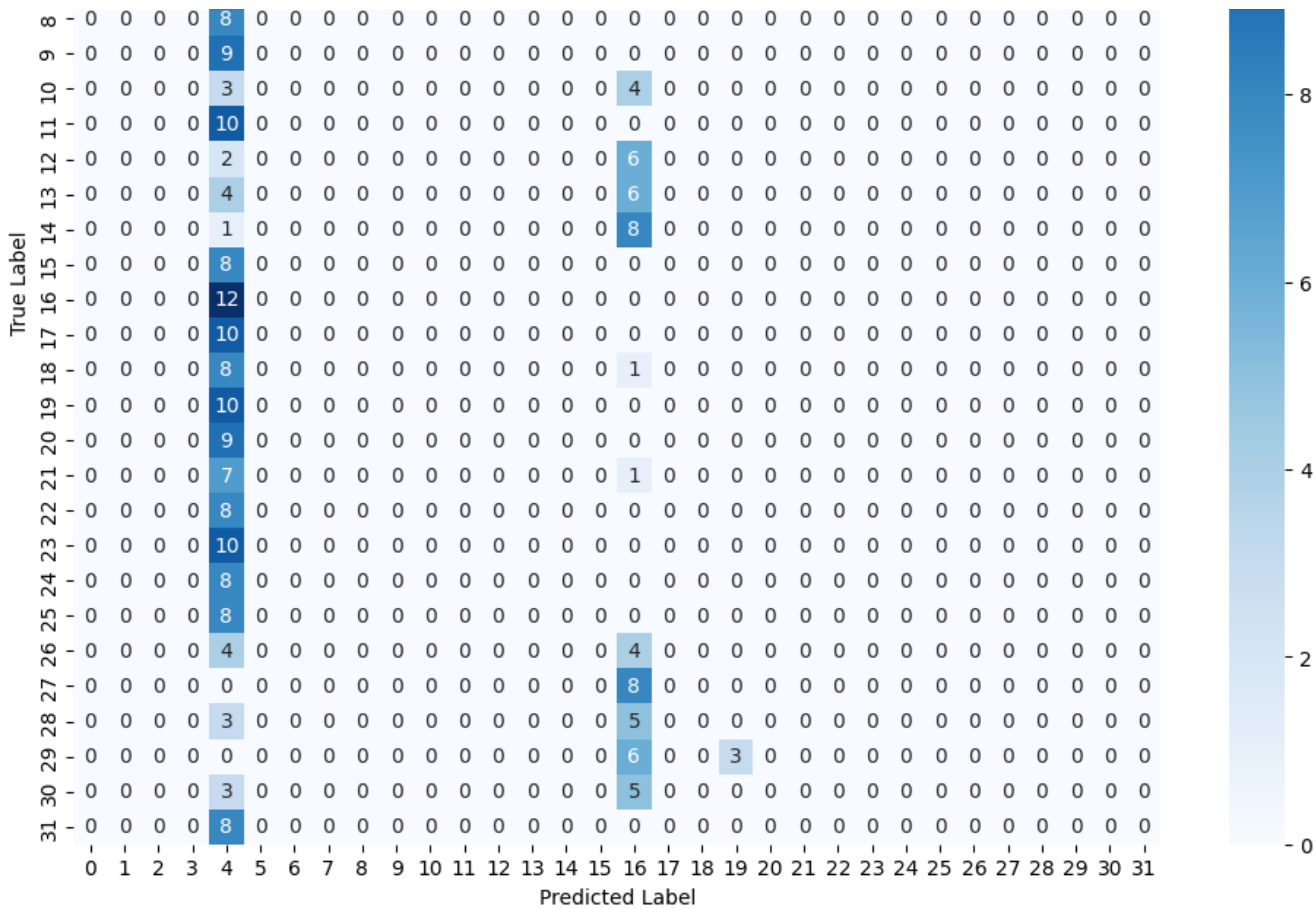

```
--- Loss Comparison ---
CNN Test Loss: 0.6464
LSTM Test Loss: 3.4529
```




```
--- LSTM Evaluation ---
Generating Classification Report and Confusion Matrix for LSTM...
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
```

```
LSTM Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         9
           1       0.00      0.00      0.00        10
           2       0.00      0.00      0.00        10
           3       0.00      0.00      0.00        11
           4       0.05      1.00      0.10        11
           5       0.00      0.00      0.00         8
           6       0.00      0.00      0.00        10
           7       0.00      0.00      0.00         8
           8       0.00      0.00      0.00         8
           9       0.00      0.00      0.00         9
          10       0.00      0.00      0.00         7
          11       0.00      0.00      0.00        10
          12       0.00      0.00      0.00         8
          13       0.00      0.00      0.00        10
          14       0.00      0.00      0.00         9
          15       0.00      0.00      0.00         8
          16       0.00      0.00      0.00        12
          17       0.00      0.00      0.00        10
          18       0.00      0.00      0.00         9
          19       0.00      0.00      0.00        10
          20       0.00      0.00      0.00         9
          21       0.00      0.00      0.00         8
          22       0.00      0.00      0.00         8
          23       0.00      0.00      0.00        10
          24       0.00      0.00      0.00         8
          25       0.00      0.00      0.00         8
          26       0.00      0.00      0.00         8
          27       0.00      0.00      0.00         8
          28       0.00      0.00      0.00         8
          29       0.00      0.00      0.00         9
          30       0.00      0.00      0.00         8
          31       0.00      0.00      0.00         8

    accuracy                           0.04       287
   macro avg       0.00      0.03      0.00       287
weighted avg       0.00      0.04      0.00       287
```




True Label
Predicted Label

```
!pip install keras-tuner
```

```
Collecting keras-tuner
  Downloading keras_tuner-1.4.7-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: keras in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-pacl
Collecting kt-legacy (from keras-tuner)
  Downloading kt_legacy-1.0.5-py3-none-any.whl.metadata (221 bytes)
Requirement already satisfied: absl-py in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: h5py in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyth
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1
Requirement already satisfied: typing-extensions>=4.6.0 in /usr/local/lib/pyth
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/pytho
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-pa
Downloading keras_tuner-1.4.7-py3-none-any.whl (129 kB)
                                          129.1/129.1 kB 4.1 MB/s eta 0:00:0
Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)
Installing collected packages: kt-legacy, keras-tuner
Successfully installed keras-tuner-1.4.7 kt-legacy-1.0.5
```

Project Conclusion - LSTM and CNN Comparision

Overall Model Performance Comparison

Metric	CNN	LSTM
Test Accuracy	**0.8780**	0.0383
Test Loss	**0.6464**	3.4529
Weighted Avg F1-Score	**0.8788**	0.0037

- The tuned **CNN model significantly outperformed** the LSTM model in all key performance metrics on the test set.
- The LSTM model struggled with classification accuracy and generalization, likely due to

the spatial nature of image data, which CNNs handle more effectively.

Insights from Classification Report & Confusion Matrix

- **Per-Class Metrics**:
  - Analyzing precision, recall, and F1-scores per class helps identify which species were well-classified and which were challenging.
- **Confusion Matrix Analysis**:
  - Provides a visual representation of which class pairs are most often misclassified.
  - **Low Precision**: Indicates other class samples are incorrectly predicted as this class.
  - **Low Recall**: Indicates this class's samples are often missed.
  - **F1-Score**: Balances both precision and recall for more complete performance insights.

Hyperparameter Tuning Insights (CNN)

The best-performing CNN architecture was discovered through systematic hyperparameter tuning:

- `Conv_1 Filters`: **96**
- `Conv_2 Filters`: **128**
- `Use Conv_3`: **True**
- `Conv_3 Filters`: **128**
- `Dense Units`: **192**
- `Dropout Rate`: **0.30**
- This configuration yielded the **highest validation accuracy** during tuning.
- The use of **early stopping** was effective in preventing overfitting for both CNN and LSTM models.

Final Summary

This project successfully implemented, trained, and evaluated both **CNN** and **LSTM** architectures for plant leaf classification using the Flavia dataset.

- The **CNN model**, specifically optimized through hyperparameter tuning, **outperformed the LSTM model** by a large margin.
- The **LSTM model**, while powerful for sequential data, was not well-suited for spatial image inputs.

Now use Transfer learning solution (PreTrained model MobileNet with additional proprietary layer

Compare between CNN, Transfer Learning with our classification layer and LSTM

```
# Transfer learning solution and CNN, Transfer Learning with our
# classification layer and LSTM comparision

import matplotlib.pyplot as plt
import numpy as np
# --- Implement Transfer Learning Solution ---

if 'X_train' in locals() and X_train.shape[0] > 0:
    print("\n--- Implementing Transfer Learning ---")


    X_train_rgb = np.repeat(X_train, 3, axis=-1)
    X_val_rgb = np.repeat(X_val, 3, axis=-1)
    X_test_rgb = np.repeat(X_test, 3, axis=-1)

    print(f"RGB Training data shape for Transfer Learning: {X_train_rgb.shape}")
    print(f"RGB Validation data shape for Transfer Learning: {X_val_rgb.shape}")
    print(f"RGB Test data shape for Transfer Learning: {X_test_rgb.shape}")


    from tensorflow.keras.applications import MobileNetV2
    from tensorflow.keras.layers import GlobalAveragePooling2D
    from tensorflow.keras.models import Model
```

```
base_model = MobileNetV2(input_shape=(LEAF_IMAGE_SIZE[0], LEAF_IMAGE_SIZE[1],
                         include_top=False,
                         weights='imagenet')

# Freeze the layers of the base model so they are not trained
# Only the new layers we add will be trained
base_model.trainable = False

# Build the new model on top of the pre-trained base
# if needed we can add few more conv2d layers.
x = base_model.output
x = GlobalAveragePooling2D()(x) # Add a Global Average Pooling layer
predictions = Dense(num_classes, activation='softmax')(x) # Add our new class

transfer_model = Model(inputs=base_model.input, outputs=predictions)

# Compile the transfer learning model
# We can use a smaller learning rate for fine-tuning later if needed
transfer_model.compile(optimizer='adam',
                       loss='categorical_crossentropy',
                       metrics=['accuracy'])

print("\nTransfer Learning Model Summary:")
transfer_model.summary()

# Train the Transfer Learning Model (Fine-tuning could be done later)
print("\nTraining Transfer Learning model...")

# Define early stopping callback (using the one defined previously)
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

transfer_history = transfer_model.fit(X_train_rgb, y_train,
                                      epochs=20, # Start with a reasonable nu
                                      batch_size=32,
                                      validation_data=(X_val_rgb, y_val),
                                      callbacks=[early_stopping])

# Evaluate the Transfer Learning Model
print("\nEvaluating Transfer Learning model on test data...")
transfer_loss, transfer_accuracy = transfer_model.evaluate(X_test_rgb, y_test
```

```
print(f"Transfer Learning Test Loss: {transfer_loss:.4f}")
print(f"Transfer Learning Test Accuracy: {transfer_accuracy:.4f}")

# Compare Accuracies with CNN and LSTM
print("\n--- Accuracy Comparison (CNN vs LSTM vs Transfer Learning) ---")
if 'accuracy' in locals() and 'lstm_accuracy' in locals():
    print(f"CNN Test Accuracy: {accuracy:.4f}")
    print(f"LSTM Test Accuracy: {lstm_accuracy:.4f}")
    print(f"Transfer Learning Test Accuracy: {transfer_accuracy:.4f}")

    labels_comp = ['CNN', 'LSTM', 'Transfer Learning']
    accuracies_comp = [accuracy, lstm_accuracy, transfer_accuracy]

    plt.figure(figsize=(8, 5))
    sns.barplot(x=labels_comp, y=accuracies_comp, palette='viridis')
    plt.ylim(0, 1)
    plt.ylabel('Test Accuracy')
    plt.title('Model Test Accuracy Comparison')
    for i, acc in enumerate(accuracies_comp):
        plt.text(i, acc + 0.02, f'{acc:.4f}', ha='center')
    plt.show()

    # You can also compare loss
    if 'loss' in locals() and 'lstm_loss' in locals():
         print("\n--- Loss Comparison (CNN vs LSTM vs Transfer Learning) ---"
         print(f"CNN Test Loss: {loss:.4f}")
         print(f"LSTM Test Loss: {lstm_loss:.4f}")
         print(f"Transfer Learning Test Loss: {transfer_loss:.4f}")

         losses_comp = [loss, lstm_loss, transfer_loss]
         plt.figure(figsize=(8, 5))
         sns.barplot(x=labels_comp, y=losses_comp, palette='magma')
         plt.ylabel('Test Loss')
         plt.title('Model Test Loss Comparison')
         for i, lss in enumerate(losses_comp):
             plt.text(i, lss + 0.01, f'{lss:.4f}', ha='center')
         plt.show()


    # Classification Report and Confusion Matrix for Transfer Learning Model
    print("\n--- Transfer Learning Evaluation ---")
    print("Generating Classification Report and Confusion Matrix for Transfer

    y_pred_transfer = transfer_model.predict(X_test_rgb)
    y_pred_classes_transfer = np.argmax(y_pred_transfer, axis=1)
```

```
            # y_true_classes is the same for all models

            print("\nTransfer Learning Classification Report:")
            if 'unique_labels' in locals() and 'filtered_target_names' in locals() an
                print(classification_report(y_true_classes, y_pred_classes_transfer,
                                            labels=present_classes_indices,
                                            target_names=filtered_target_names,
                                            zero_division=0))

                conf_matrix_transfer = confusion_matrix(y_true_classes, y_pred_classe

                plt.figure(figsize=(18, 16))
                sns.heatmap(conf_matrix_transfer, annot=True, fmt='d', cmap='Blues',
                            xticklabels=filtered_target_names,
                            yticklabels=filtered_target_names)
                plt.xlabel('Predicted Label')
                plt.ylabel('True Label')
                plt.title('Transfer Learning Confusion Matrix')
                plt.tight_layout()
                plt.show()
            else:
                 print("Cannot generate detailed report and confusion matrix for Tran


        else:
            print("CNN and/or LSTM accuracy not found. Ensure previous model evaluati

    else:
        print("Training data (X_train, y_train) is not available. Cannot build and tr
        print("Please ensure the data preprocessing steps ran correctly.")
```

```
--- Implementing Transfer Learning ---
RGB Training data shape for Transfer Learning: (1334, 128, 128, 3)
RGB Validation data shape for Transfer Learning: (286, 128, 128, 3)
RGB Test data shape for Transfer Learning: (287, 128, 128, 3)
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicat
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step

Transfer Learning Model Summary:
```

Model: "functional_11"

Layer (type)	Output Shape	Param #	Connected to
input_layer_2 (InputLayer)	(None, 128, 128, 3)	0	-

Conv1 (Conv2D)	(None, 64, 64, 32)	864	input_layer_2[0]…
bn_Conv1 (BatchNormalizatio…	(None, 64, 64, 32)	128	Conv1[0][0]
Conv1_relu (ReLU)	(None, 64, 64, 32)	0	bn_Conv1[0][0]
expanded_conv_dept… (DepthwiseConv2D)	(None, 64, 64, 32)	288	Conv1_relu[0][0]
expanded_conv_dept… (BatchNormalizatio…	(None, 64, 64, 32)	128	expanded_conv_de…
expanded_conv_dept… (ReLU)	(None, 64, 64, 32)	0	expanded_conv_de…
expanded_conv_proj… (Conv2D)	(None, 64, 64, 16)	512	expanded_conv_de…
expanded_conv_proj… (BatchNormalizatio…	(None, 64, 64, 16)	64	expanded_conv_pr…
block_1_expand (Conv2D)	(None, 64, 64, 96)	1,536	expanded_conv_pr…
block_1_expand_BN (BatchNormalizatio…	(None, 64, 64, 96)	384	block_1_expand[0…
block_1_expand_relu (ReLU)	(None, 64, 64, 96)	0	block_1_expand_B…
block_1_pad (ZeroPadding2D)	(None, 65, 65, 96)	0	block_1_expand_r…
block_1_depthwise (DepthwiseConv2D)	(None, 32, 32, 96)	864	block_1_pad[0][0]
block_1_depthwise_… (BatchNormalizatio…	(None, 32, 32, 96)	384	block_1_depthwis…
block_1_depthwise_… (ReLU)	(None, 32, 32, 96)	0	block_1_depthwis…
block_1_project (Conv2D)	(None, 32, 32, 24)	2,304	block_1_depthwis…
block_1_project_BN (BatchNormalizatio…	(None, 32, 32, 24)	96	block_1_project[…
block_2_expand (Conv2D)	(None, 32, 32, 144)	3,456	block_1_project_…

(Conv2D)	144)		
block_2_expand_BN (BatchNormalizatio…	(None, 32, 32, 144)	576	block_2_expand[0…
block_2_expand_relu (ReLU)	(None, 32, 32, 144)	0	block_2_expand_B…
block_2_depthwise (DepthwiseConv2D)	(None, 32, 32, 144)	1,296	block_2_expand_r…
block_2_depthwise_… (BatchNormalizatio…	(None, 32, 32, 144)	576	block_2_depthwis…
block_2_depthwise_… (ReLU)	(None, 32, 32, 144)	0	block_2_depthwis…
block_2_project (Conv2D)	(None, 32, 32, 24)	3,456	block_2_depthwis…
block_2_project_BN (BatchNormalizatio…	(None, 32, 32, 24)	96	block_2_project[…
block_2_add (Add)	(None, 32, 32, 24)	0	block_1_project_… block_2_project_…
block_3_expand (Conv2D)	(None, 32, 32, 144)	3,456	block_2_add[0][0]
block_3_expand_BN (BatchNormalizatio…	(None, 32, 32, 144)	576	block_3_expand[0…
block_3_expand_relu (ReLU)	(None, 32, 32, 144)	0	block_3_expand_B…
block_3_pad (ZeroPadding2D)	(None, 33, 33, 144)	0	block_3_expand_r…
block_3_depthwise (DepthwiseConv2D)	(None, 16, 16, 144)	1,296	block_3_pad[0][0]
block_3_depthwise_… (BatchNormalizatio…	(None, 16, 16, 144)	576	block_3_depthwis…
block_3_depthwise_… (ReLU)	(None, 16, 16, 144)	0	block_3_depthwis…
block_3_project (Conv2D)	(None, 16, 16, 32)	4,608	block_3_depthwis…
block_3_project_BN (BatchNormalizatio…	(None, 16, 16, 32)	128	block_3_project[…
block_4_expand	(None, 16, 16,	6,144	block_3_project_…

(Conv2D)	192)		
block_4_expand_BN (BatchNormalizatio…	(None, 16, 16, 192)	768	block_4_expand[0…
block_4_expand_relu (ReLU)	(None, 16, 16, 192)	0	block_4_expand_B…
block_4_depthwise (DepthwiseConv2D)	(None, 16, 16, 192)	1,728	block_4_expand_r…
block_4_depthwise_… (BatchNormalizatio…	(None, 16, 16, 192)	768	block_4_depthwis…
block_4_depthwise_… (ReLU)	(None, 16, 16, 192)	0	block_4_depthwis…
block_4_project (Conv2D)	(None, 16, 16, 32)	6,144	block_4_depthwis…
block_4_project_BN (BatchNormalizatio…	(None, 16, 16, 32)	128	block_4_project[…
block_4_add (Add)	(None, 16, 16, 32)	0	block_3_project_… block_4_project_…
block_5_expand (Conv2D)	(None, 16, 16, 192)	6,144	block_4_add[0][0]
block_5_expand_BN (BatchNormalizatio…	(None, 16, 16, 192)	768	block_5_expand[0…
block_5_expand_relu (ReLU)	(None, 16, 16, 192)	0	block_5_expand_B…
block_5_depthwise (DepthwiseConv2D)	(None, 16, 16, 192)	1,728	block_5_expand_r…
block_5_depthwise_… (BatchNormalizatio…	(None, 16, 16, 192)	768	block_5_depthwis…
block_5_depthwise_… (ReLU)	(None, 16, 16, 192)	0	block_5_depthwis…
block_5_project (Conv2D)	(None, 16, 16, 32)	6,144	block_5_depthwis…
block_5_project_BN (BatchNormalizatio…	(None, 16, 16, 32)	128	block_5_project[…
block_5_add (Add)	(None, 16, 16, 32)	0	block_4_add[0][0… block_5_project_…
block_6_expand	(None, 16, 16,	6,144	block_5_add[0][0]

block_6_expand (Conv2D)	(None, 16, 16, 192)	6,144	block_5_add[0][0]
block_6_expand_BN (BatchNormalizatio…	(None, 16, 16, 192)	768	block_6_expand[0…
block_6_expand_relu (ReLU)	(None, 16, 16, 192)	0	block_6_expand_B…
block_6_pad (ZeroPadding2D)	(None, 17, 17, 192)	0	block_6_expand_r…
block_6_depthwise (DepthwiseConv2D)	(None, 8, 8, 192)	1,728	block_6_pad[0][0]
block_6_depthwise_… (BatchNormalizatio…	(None, 8, 8, 192)	768	block_6_depthwis…
block_6_depthwise_… (ReLU)	(None, 8, 8, 192)	0	block_6_depthwis…
block_6_project (Conv2D)	(None, 8, 8, 64)	12,288	block_6_depthwis…
block_6_project_BN (BatchNormalizatio…	(None, 8, 8, 64)	256	block_6_project[…
block_7_expand (Conv2D)	(None, 8, 8, 384)	24,576	block_6_project_…
block_7_expand_BN (BatchNormalizatio…	(None, 8, 8, 384)	1,536	block_7_expand[0…
block_7_expand_relu (ReLU)	(None, 8, 8, 384)	0	block_7_expand_B…
block_7_depthwise (DepthwiseConv2D)	(None, 8, 8, 384)	3,456	block_7_expand_r…
block_7_depthwise_… (BatchNormalizatio…	(None, 8, 8, 384)	1,536	block_7_depthwis…
block_7_depthwise_… (ReLU)	(None, 8, 8, 384)	0	block_7_depthwis…
block_7_project (Conv2D)	(None, 8, 8, 64)	24,576	block_7_depthwis…
block_7_project_BN (BatchNormalizatio…	(None, 8, 8, 64)	256	block_7_project[…
block_7_add (Add)	(None, 8, 8, 64)	0	block_6_project_… block_7_project_…

block_8_expand (Conv2D)	(None, 8, 8, 384)	24,576	block_7_add[0][0]
block_8_expand_BN (BatchNormalizatio…	(None, 8, 8, 384)	1,536	block_8_expand[0…
block_8_expand_relu (ReLU)	(None, 8, 8, 384)	0	block_8_expand_B…
block_8_depthwise (DepthwiseConv2D)	(None, 8, 8, 384)	3,456	block_8_expand_r…
block_8_depthwise_… (BatchNormalizatio…	(None, 8, 8, 384)	1,536	block_8_depthwis…
block_8_depthwise_… (ReLU)	(None, 8, 8, 384)	0	block_8_depthwis…
block_8_project (Conv2D)	(None, 8, 8, 64)	24,576	block_8_depthwis…
block_8_project_BN (BatchNormalizatio…	(None, 8, 8, 64)	256	block_8_project[…
block_8_add (Add)	(None, 8, 8, 64)	0	block_7_add[0][0… block_8_project_…
block_9_expand (Conv2D)	(None, 8, 8, 384)	24,576	block_8_add[0][0]
block_9_expand_BN (BatchNormalizatio…	(None, 8, 8, 384)	1,536	block_9_expand[0…
block_9_expand_relu (ReLU)	(None, 8, 8, 384)	0	block_9_expand_B…
block_9_depthwise (DepthwiseConv2D)	(None, 8, 8, 384)	3,456	block_9_expand_r…
block_9_depthwise_… (BatchNormalizatio…	(None, 8, 8, 384)	1,536	block_9_depthwis…
block_9_depthwise_… (ReLU)	(None, 8, 8, 384)	0	block_9_depthwis…
block_9_project (Conv2D)	(None, 8, 8, 64)	24,576	block_9_depthwis…
block_9_project_BN (BatchNormalizatio…	(None, 8, 8, 64)	256	block_9_project[…
block_9_add (Add)	(None, 8, 8, 64)	0	block_8_add[0][0… block_9_project_…

block_10_expand (Conv2D)	(None, 8, 8, 384)	24,576	block_9_add[0][0]
block_10_expand_BN (BatchNormalizatio…	(None, 8, 8, 384)	1,536	block_10_expand[…
block_10_expand_re… (ReLU)	(None, 8, 8, 384)	0	block_10_expand_…
block_10_depthwise (DepthwiseConv2D)	(None, 8, 8, 384)	3,456	block_10_expand_…
block_10_depthwise… (BatchNormalizatio…	(None, 8, 8, 384)	1,536	block_10_depthwi…
block_10_depthwise… (ReLU)	(None, 8, 8, 384)	0	block_10_depthwi…
block_10_project (Conv2D)	(None, 8, 8, 96)	36,864	block_10_depthwi…
block_10_project_BN (BatchNormalizatio…	(None, 8, 8, 96)	384	block_10_project…
block_11_expand (Conv2D)	(None, 8, 8, 576)	55,296	block_10_project…
block_11_expand_BN (BatchNormalizatio…	(None, 8, 8, 576)	2,304	block_11_expand[…
block_11_expand_re… (ReLU)	(None, 8, 8, 576)	0	block_11_expand_…
block_11_depthwise (DepthwiseConv2D)	(None, 8, 8, 576)	5,184	block_11_expand_…
block_11_depthwise… (BatchNormalizatio…	(None, 8, 8, 576)	2,304	block_11_depthwi…
block_11_depthwise… (ReLU)	(None, 8, 8, 576)	0	block_11_depthwi…
block_11_project (Conv2D)	(None, 8, 8, 96)	55,296	block_11_depthwi…
block_11_project_BN (BatchNormalizatio…	(None, 8, 8, 96)	384	block_11_project…
block_11_add (Add)	(None, 8, 8, 96)	0	block_10_project… block_11_project…
block_12_expand (Conv2D)	(None, 8, 8, 576)	55,296	block_11_add[0][…

block_12_expand_BN (BatchNormalizatio…	(None, 8, 8, 576)	2,304	block_12_expand[…
block_12_expand_re… (ReLU)	(None, 8, 8, 576)	0	block_12_expand_…
block_12_depthwise (DepthwiseConv2D)	(None, 8, 8, 576)	5,184	block_12_expand_…
block_12_depthwise… (BatchNormalizatio…	(None, 8, 8, 576)	2,304	block_12_depthwi…
block_12_depthwise… (ReLU)	(None, 8, 8, 576)	0	block_12_depthwi…
block_12_project (Conv2D)	(None, 8, 8, 96)	55,296	block_12_depthwi…
block_12_project_BN (BatchNormalizatio…	(None, 8, 8, 96)	384	block_12_project…
block_12_add (Add)	(None, 8, 8, 96)	0	block_11_add[0][… block_12_project…
block_13_expand (Conv2D)	(None, 8, 8, 576)	55,296	block_12_add[0][…
block_13_expand_BN (BatchNormalizatio…	(None, 8, 8, 576)	2,304	block_13_expand[…
block_13_expand_re… (ReLU)	(None, 8, 8, 576)	0	block_13_expand_…
block_13_pad (ZeroPadding2D)	(None, 9, 9, 576)	0	block_13_expand_…
block_13_depthwise (DepthwiseConv2D)	(None, 4, 4, 576)	5,184	block_13_pad[0][…
block_13_depthwise… (BatchNormalizatio…	(None, 4, 4, 576)	2,304	block_13_depthwi…
block_13_depthwise… (ReLU)	(None, 4, 4, 576)	0	block_13_depthwi…
block_13_project (Conv2D)	(None, 4, 4, 160)	92,160	block_13_depthwi…
block_13_project_BN (BatchNormalizatio…	(None, 4, 4, 160)	640	block_13_project…
block_14_expand (Conv2D)	(None, 4, 4, 960)	153,600	block_13_project…

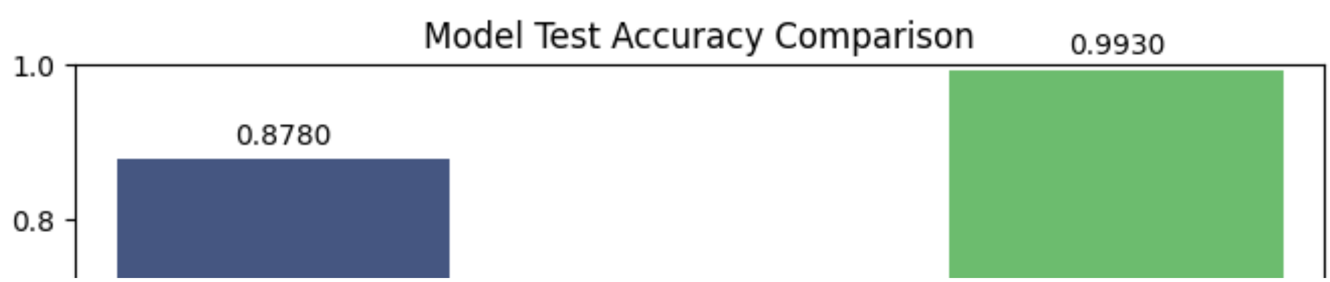
block_14_expand_BN (BatchNormalizatio…	(None, 4, 4, 960)	3,840	block_14_expand[…
block_14_expand_re… (ReLU)	(None, 4, 4, 960)	0	block_14_expand_…
block_14_depthwise (DepthwiseConv2D)	(None, 4, 4, 960)	8,640	block_14_expand_…
block_14_depthwise… (BatchNormalizatio…	(None, 4, 4, 960)	3,840	block_14_depthwi…
block_14_depthwise… (ReLU)	(None, 4, 4, 960)	0	block_14_depthwi…
block_14_project (Conv2D)	(None, 4, 4, 160)	153,600	block_14_depthwi…
block_14_project_BN (BatchNormalizatio…	(None, 4, 4, 160)	640	block_14_project…
block_14_add (Add)	(None, 4, 4, 160)	0	block_13_project… block_14_project…
block_15_expand (Conv2D)	(None, 4, 4, 960)	153,600	block_14_add[0][…
block_15_expand_BN (BatchNormalizatio…	(None, 4, 4, 960)	3,840	block_15_expand[…
block_15_expand_re… (ReLU)	(None, 4, 4, 960)	0	block_15_expand_…
block_15_depthwise (DepthwiseConv2D)	(None, 4, 4, 960)	8,640	block_15_expand_…
block_15_depthwise… (BatchNormalizatio…	(None, 4, 4, 960)	3,840	block_15_depthwi…
block_15_depthwise… (ReLU)	(None, 4, 4, 960)	0	block_15_depthwi…
block_15_project (Conv2D)	(None, 4, 4, 160)	153,600	block_15_depthwi…
block_15_project_BN (BatchNormalizatio…	(None, 4, 4, 160)	640	block_15_project…
block_15_add (Add)	(None, 4, 4, 160)	0	block_14_add[0][… block_15_project…
block_16_expand (Conv2D)	(None, 4, 4, 960)	153,600	block_15_add[0][…

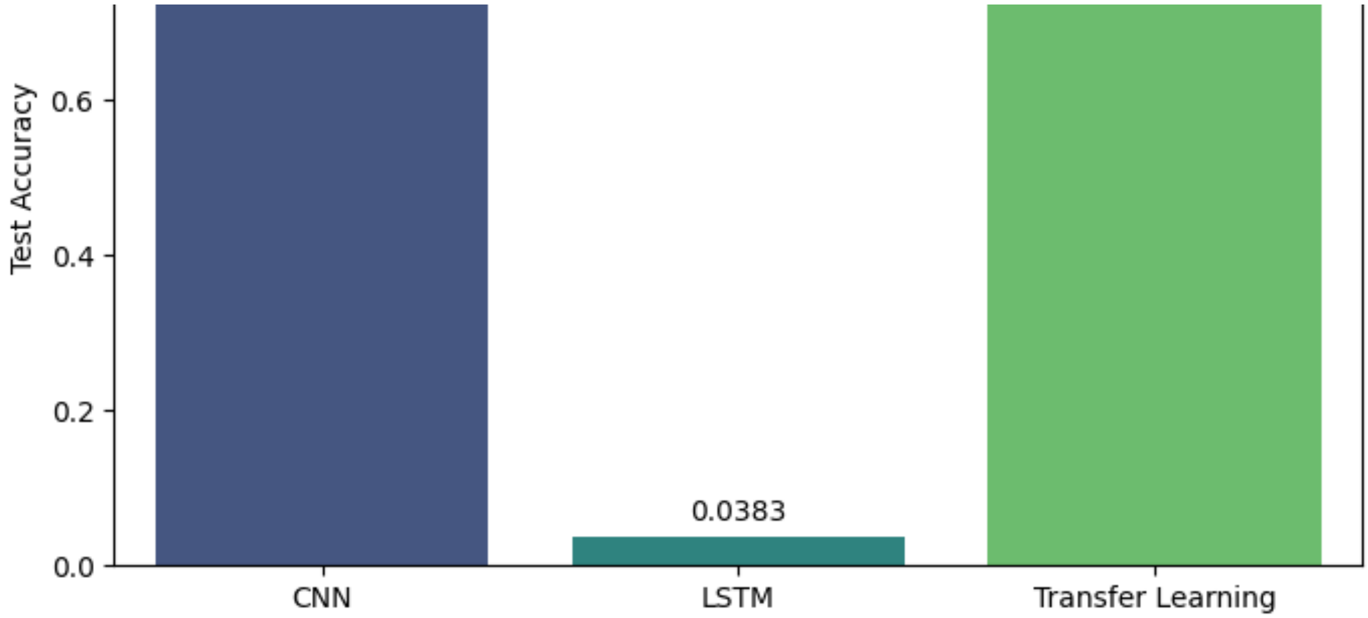
(Conv2D)			
block_16_expand_BN (BatchNormalizatio…	(None, 4, 4, 960)	3,840	block_16_expand[…
block_16_expand_re… (ReLU)	(None, 4, 4, 960)	0	block_16_expand_…
block_16_depthwise (DepthwiseConv2D)	(None, 4, 4, 960)	8,640	block_16_expand_…
block_16_depthwise… (BatchNormalizatio…	(None, 4, 4, 960)	3,840	block_16_depthwi…
block_16_depthwise… (ReLU)	(None, 4, 4, 960)	0	block_16_depthwi…
block_16_project (Conv2D)	(None, 4, 4, 320)	307,200	block_16_depthwi…
block_16_project_BN (BatchNormalizatio…	(None, 4, 4, 320)	1,280	block_16_project…
Conv_1 (Conv2D)	(None, 4, 4, 1280)	409,600	block_16_project…
Conv_1_bn (BatchNormalizatio…	(None, 4, 4, 1280)	5,120	Conv_1[0][0]
out_relu (ReLU)	(None, 4, 4, 1280)	0	Conv_1_bn[0][0]
global_average_poo… (GlobalAveragePool…	(None, 1280)	0	out_relu[0][0]
dense_3 (Dense)	(None, 32)	40,992	global_average_p…

Total params: 2,298,976 (8.77 MB)
Trainable params: 40,992 (160.12 KB)
Non-trainable params: 2,257,984 (8.61 MB)

```
Training Transfer Learning model...
Epoch 1/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 20s 268ms/step - accuracy: 0.3789 - loss: 2.6536 -
Epoch 2/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9477 - loss: 0.4005 - va
Epoch 3/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9761 - loss: 0.1792 - va
Epoch 4/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9866 - loss: 0.1216 - va
Epoch 5/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9925 - loss: 0.0897 - va
Epoch 6/20
```

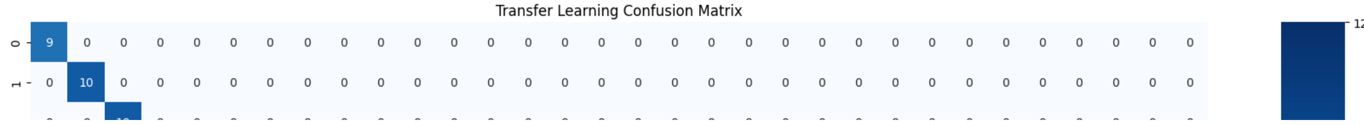
```
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9953 - loss: 0.0607 - va
Epoch 7/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9971 - loss: 0.0506 - va
Epoch 8/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9993 - loss: 0.0392 - va
Epoch 9/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 1.0000 - loss: 0.0320 - va
Epoch 10/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 1.0000 - loss: 0.0265 - va
Epoch 11/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 1.0000 - loss: 0.0247 - va
Epoch 12/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 1.0000 - loss: 0.0199 - va
Epoch 13/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 1.0000 - loss: 0.0181 - va
Epoch 14/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 1.0000 - loss: 0.0157 - va
Epoch 15/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 1.0000 - loss: 0.0139 - va
Epoch 16/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 1.0000 - loss: 0.0124 - va
Epoch 17/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 1.0000 - loss: 0.0115 - va
Epoch 18/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 1.0000 - loss: 0.0113 - va
Epoch 19/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 1.0000 - loss: 0.0096 - va
Epoch 20/20
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 1.0000 - loss: 0.0099 - va

Evaluating Transfer Learning model on test data...
Transfer Learning Test Loss: 0.0568
Transfer Learning Test Accuracy: 0.9930

--- Accuracy Comparison (CNN vs LSTM vs Transfer Learning) ---
CNN Test Accuracy: 0.8780
LSTM Test Accuracy: 0.0383
Transfer Learning Test Accuracy: 0.9930
/tmp/ipython-input-17-585755722.py:97: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed ir

  sns.barplot(x=labels_comp, y=accuracies_comp, palette='viridis')
```

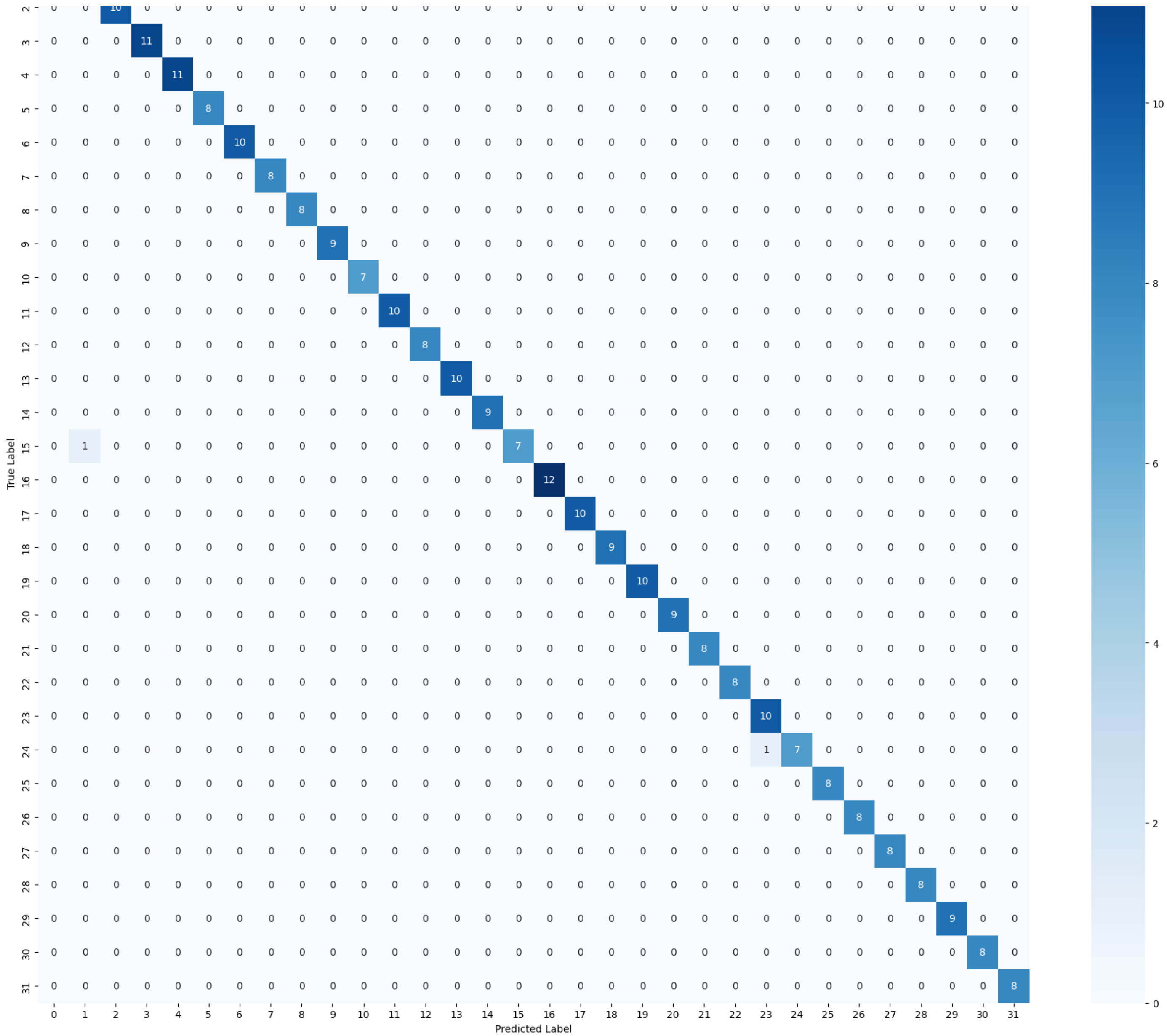





```
--- Loss Comparison (CNN vs LSTM vs Transfer Learning) ---
CNN Test Loss: 0.6464
LSTM Test Loss: 3.4529
Transfer Learning Test Loss: 0.0568
/tmp/ipython-input-17-585755722.py:114: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed ir

  sns.barplot(x=labels_comp, y=losses_comp, palette='magma')
```


```
--- Transfer Learning Evaluation ---
Generating Classification Report and Confusion Matrix for Transfer Learning Mc
9/9 ━━━━━━━━━━━━━━━━━━━━ 7s 443ms/step

Transfer Learning Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       0.91      1.00      0.95        10
           2       1.00      1.00      1.00        10
           3       1.00      1.00      1.00        11
           4       1.00      1.00      1.00        11
           5       1.00      1.00      1.00         8
           6       1.00      1.00      1.00        10
           7       1.00      1.00      1.00         8
           8       1.00      1.00      1.00         8
           9       1.00      1.00      1.00         9
          10       1.00      1.00      1.00         7
          11       1.00      1.00      1.00        10
          12       1.00      1.00      1.00         8
          13       1.00      1.00      1.00        10
          14       1.00      1.00      1.00         9
          15       1.00      0.88      0.93         8
          16       1.00      1.00      1.00        12
          17       1.00      1.00      1.00        10
          18       1.00      1.00      1.00         9
          19       1.00      1.00      1.00        10
          20       1.00      1.00      1.00         9
          21       1.00      1.00      1.00         8
          22       1.00      1.00      1.00         8
          23       0.91      1.00      0.95        10
          24       1.00      0.88      0.93         8
          25       1.00      1.00      1.00         8
          26       1.00      1.00      1.00         8
          27       1.00      1.00      1.00         8
          28       1.00      1.00      1.00         8
          29       1.00      1.00      1.00         9
          30       1.00      1.00      1.00         8
          31       1.00      1.00      1.00         8

    accuracy                           0.99       287
   macro avg       0.99      0.99      0.99       287
weighted avg       0.99      0.99      0.99       287
```




True Label
Predicted Label

#

Predict with Transfer learning model(MobileNetV2)

```
#

import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.applications import MobileNetV2 # Import MobileNetV2


# Check if the transfer_model and test data are available
if 'transfer_model' in locals() and 'X_test_rgb' in locals() and 'y_test' in loca
    print("\n--- Predicting and Plotting Samples with Transfer Learning Model ---

    num_samples_to_plot = 5 # Number of random samples to plot
    sample_indices = random.sample(range(X_test_rgb.shape[0]), min(num_samples_to

    plt.figure(figsize=(15, 5 * ((num_samples_to_plot + 4) // 5))) # Adjust figur

    for i, sample_index in enumerate(sample_indices):
        # Get the sample image and true label
        sample_image = X_test_rgb[sample_index]
        true_label_encoded = y_test[sample_index]
        true_label_index = np.argmax(true_label_encoded)

        # Predict the class for the sample image
        sample_image_input = np.expand_dims(sample_image, axis=0) # Add batch dim
        prediction_probs = transfer_model.predict(sample_image_input)
        predicted_label_index = np.argmax(prediction_probs)
```

```
        confidence = np.max(prediction_probs) * 100

        # Get the original species label from the unique_labels list
        # Ensure unique_labels is available
        if 'unique_labels' in locals():
            true_species = unique_labels[true_label_index]
            predicted_species = unique_labels[predicted_label_index]
        else:
            true_species = f"Class {true_label_index}"
            predicted_species = f"Class {predicted_label_index}"
            print("Warning: unique_labels not found, using integer indices for sp


        display_image = (sample_image[:,:,0] * 255).astype(np.uint8)

        plt.subplot(1, num_samples_to_plot, i + 1) # Arrange plots in a row
        plt.imshow(display_image, cmap='gray')
        plt.title(f"True: {true_species}\nPred: {predicted_species} ({confidence:
        plt.axis('off')

    plt.tight_layout()
    plt.show()

else:
    print("Transfer learning model or test data not available. Cannot predict and
    print(f"Is transfer_model defined: {'transfer_model' in locals()}")
    print(f"Is X_test_rgb defined: {'X_test_rgb' in locals()}")
    print(f"Is y_test defined: {'y_test' in locals()}")
    if 'X_test_rgb' in locals():
        print(f"X_test_rgb shape: {X_test_rgb.shape}")
    if 'y_test' in locals():
         print(f"y_test shape: {y_test.shape}")
```

```
--- Predicting and Plotting Samples with Transfer Learning Model ---
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
```

Conclusion: Implementing Transfer Learning for Leaf Classification

Transfer learning was applied using **MobileNetV2** as the base model to improve classification performance on the Flavia leaf dataset.

Input Data Configuration for Transfer Learning

Dataset	Shape
Training Set	(1334, 128, 128, 3)
Validation Set	(286, 128, 128, 3)
Test Set	(287, 128, 128, 3)

- All images were converted to **RGB** and resized to **128×128×3**.
- **MobileNetV2 weights** were loaded from TensorFlow's Keras application hub (excluding top layers).

Transfer Learning Model Summary

- Base model: **MobileNetV2** (frozen layers)
- Top layers added:
  - `GlobalAveragePooling2D`
  - `Dense(32, activation='softmax')`

Parameters	Count
Total Parameters	2,298,976
Trainable Parameters	40,992
Non-Trainable Parameters	2,257,984

Training Results

The model was trained for **20 epochs** with **early stopping** based on validation loss.

Key Training Metrics (Final Epoch):

- **Training Accuracy**: 100%
- **Validation Accuracy**: 98.6%
- **Validation Loss**: 0.0514

Final Test Performance

Metric	Value
Test Accuracy	**99.3%**
Test Loss	0.0568

Model Comparison Summary

Model	Test Accuracy	Test Loss
CNN (custom)	87.8%	0.6464
LSTM	3.8%	3.4529
Transfer Learning (MobileNetV2)	**99.3%**	**0.0568**

> Transfer learning outperformed both CNN and LSTM by a significant margin.

Transfer Learning Classification Report

- **Overall Accuracy**: 99.3%
- **Macro Average F1-Score**: 0.99
- **Weighted Average F1-Score**: 0.99
- Most classes achieved perfect precision, recall, and F1-score (1.00).
- A few classes had slight misclassifications with F1-scores around 0.93–0.95.

Notable Misclassifications:

- Class 15: F1-score = 0.93
- Class 24: F1-score = 0.93
- Class 1 and Class 23: F1-score = 0.95

These may indicate subtle leaf variations or overlaps in morphological features.

Key Takeaways

- **Transfer Learning with MobileNetV2** dramatically improved performance while requiring fewer trainable parameters.
- Even with limited training data (~1,300 images), leveraging a pretrained network proved highly effective.
- The **LSTM model** was not suitable for spatial image data and severely underperformed.
- **CNN performed decently**, but fell short compared to the transfer learning approach.

END

END