from google.colab import drive

drive.mount('/content/drive')

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import DenseNet121

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.optimizers import Adam

# Define the path to your dataset directory

dataset\_directory = '/content/drive/MyDrive/dataset'

# Define the number of classes (weed species)

num\_classes = 4  # Adjust this based on your dataset

# Set hyperparameters

input\_shape = (128, 128, 3)

batch\_size = 32

epochs = 30

# Data augmentation and preprocessing

datagen = ImageDataGenerator(

    rescale=1.0/255.0,

    rotation\_range=20,

    width\_shift\_range=0.2,

    height\_shift\_range=0.2,

    shear\_range=0.2,

    zoom\_range=0.2,

    horizontal\_flip=True,

    fill\_mode='nearest',

    validation\_split=0.2  # Adjust the validation split

)

train\_generator = datagen.flow\_from\_directory(

    dataset\_directory,

    target\_size=input\_shape[:2],

    batch\_size=batch\_size,

    class\_mode='categorical',

    subset='training'

)

validation\_generator = datagen.flow\_from\_directory(

    dataset\_directory,

    target\_size=input\_shape[:2],

    batch\_size=batch\_size,

    class\_mode='categorical',

    subset='validation'

)

# Load the pre-trained DenseNet121 model

base\_model = DenseNet121(weights='imagenet', include\_top=False, input\_shape=input\_shape)

# Build your custom model on top of DenseNet121

model = Sequential([

    base\_model,

    GlobalAveragePooling2D(),

    Dense(128, activation='relu'),

    Dropout(0.5),

    Dense(num\_classes, activation='softmax')

])

# Specify which layers to fine-tune (unfreeze)

fine\_tune\_at = 100  # You can adjust this number based on your specific requirements

for layer in base\_model.layers[:fine\_tune\_at]:

    layer.trainable = False

# Compile the model

model.compile(optimizer=Adam(learning\_rate=0.001), loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(

    train\_generator,

    steps\_per\_epoch=len(train\_generator),

    epochs=epochs,

    validation\_data=validation\_generator,

    validation\_steps=len(validation\_generator)

)

# Save the trained model

model.save('densenet121\_fine\_tuned\_weed\_classifier\_model.h5')

# Evaluate on validation data

validation\_results = model.evaluate(validation\_generator)

# Evaluate on training data

training\_results = model.evaluate(train\_generator)

print("Training Loss:", training\_results[0])

print("Training Accuracy:", training\_results[1])

print("Validation Loss:", validation\_results[0])

print("Validation Accuracy:", validation\_results[1])

import matplotlib.pyplot as plt

# Plot training & validation accuracy values

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight\_layout()

plt.show()

import numpy as np

from sklearn.metrics import confusion\_matrix, classification\_report

import seaborn as sns

import matplotlib.pyplot as plt

# Load the saved model (if not already loaded)

# For example:

# from tensorflow.keras.models import load\_model

# loaded\_model = load\_model('weed\_classifier\_model.h5')

# Load the validation data for prediction

validation\_generator = datagen.flow\_from\_directory(

    dataset\_directory,

    target\_size=input\_shape[:2],

    batch\_size=batch\_size,

    class\_mode='categorical',

    subset='validation',  # Use the validation subset for testing

    shuffle=False  # Important: Set shuffle to False to ensure predictions match the true labels

)

# Perform predictions on the validation data

predictions = model.predict(validation\_generator)

# Get the predicted labels

y\_pred = np.argmax(predictions, axis=1)

# Get the true labels directly from validation\_generator

y\_true = validation\_generator.classes

# Step 9: Print the confusion matrix and classification report

cm = confusion\_matrix(y\_true, y\_pred)

print("Confusion Matrix:")

print(cm)

target\_names = ['broadleaf', 'grass', 'soil', 'soybean']  # Add class names accordingly

classification\_rep = classification\_report(y\_true, y\_pred, target\_names=target\_names)

print("Classification Report:")

print(classification\_rep)

# Step 10: Plot the confusion matrix with custom colors

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt="d", cmap='Oranges', xticklabels=target\_names, yticklabels=target\_names)

plt.title("Confusion Matrix")

plt.xlabel("Predicted Labels")

plt.ylabel("True Labels")

plt.show()

# Step 11: Calculate evaluation metrics

TN = cm[0, 0]

FP = cm[0, 1] + cm[0, 2] + cm[0, 3]

FN = cm[1, 0] + cm[2, 0] + cm[3, 0]

TP = cm[1, 1] + cm[1, 2] + cm[1, 3] + cm[2, 1] + cm[2, 2] + cm[2, 3] + cm[3, 1] + cm[3, 2] + cm[3, 3]

precision = TP / (TP + FP)

recall = TP / (TP + FN)

sensitivity = recall

specificity = TN / (TN + FP)

f1\_score = 2 \* (precision \* recall) / (precision + recall)

informedness = sensitivity + specificity - 1

gmean = np.sqrt(sensitivity \* specificity)

omission\_error = FN / (FN + TP)

commission\_error = FP / (FP + TN)

user\_accuracy = TP / (TP + FN)

producer\_accuracy = TP / (TP + FP)

overall\_accuracy = (TP + TN) / (TP + TN + FP + FN)

kappa = (overall\_accuracy - (FP + FN) / (TP + TN + FP + FN)) / (1 - (FP + FN) / (TP + TN + FP + FN))

# Step 12: Print the evaluation metrics

print("Precision: {:.4f}".format(precision))

print("Recall: {:.4f}".format(recall))

print("Sensitivity: {:.4f}".format(sensitivity))

print("Specificity: {:.4f}".format(specificity))

print("F1 Score: {:.4f}".format(f1\_score))

print("Informedness (IBM): {:.4f}".format(informedness))

print("Geometric Mean (G-mean): {:.4f}".format(gmean))

print("Omission Error: {:.4f}".format(omission\_error))

print("Commission Error: {:.4f}".format(commission\_error))

print("User Accuracy: {:.4f}".format(user\_accuracy))

print("Producer Accuracy: {:.4f}".format(producer\_accuracy))

print("Kappa Coefficient: {:.4f}".format(kappa))

print("Overall Accuracy: {:.4f}".format(overall\_accuracy))