import tensorflow as tf

import matplotlib.pyplot as plt

import numpy as np

from keras.preprocessing.image import ImageDataGenerator

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

import numpy as np

# Define paths to your dataset

train\_data\_dir = '/content/drive/MyDrive/waste project/data/train'

val\_data\_dir = '/content/drive/MyDrive/waste project/data/valid'

test\_data\_dir = '/content/drive/MyDrive/waste project/data/test'

# Image data generators for training and validation data

train\_datagen = ImageDataGenerator(

rescale=1./255,

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'

)

val\_datagen = ImageDataGenerator(rescale=1./255)

# Define batch size

batch\_size = 32

# Load and preprocess training data

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

train\_data\_dir,

target\_size=(224, 224), # MobileNetV2 input size

batch\_size=batch\_size,

class\_mode='categorical'

)

# Load and preprocess validation data

validation\_generator = val\_datagen.flow\_from\_directory(

val\_data\_dir,

target\_size=(224, 224), # MobileNetV2 input size

batch\_size=batch\_size,

class\_mode='categorical'

)

test\_generator = val\_datagen.flow\_from\_directory(

test\_data\_dir,

target\_size=(224, 224),

batch\_size=batch\_size,

class\_mode='categorical',

shuffle=False # Important: set shuffle to False

)

# Load MobileNetV2 pre-trained on ImageNet without the top classification layer

base\_model = tf.keras.applications.MobileNetV2(

weights='imagenet',

include\_top=False,

input\_shape=(224, 224, 3)

)

# Freeze the pre-trained layers

base\_model.trainable = False

# Add custom classification head on top of MobileNetV2 base

model = tf.keras.Sequential([

base\_model,

tf.keras.layers.GlobalAveragePooling2D(),

tf.keras.layers.Dense(256, activation='relu'),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Dense(6, activation='softmax') # 6 classes for waste classification

])

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(

train\_generator,

steps\_per\_epoch=train\_generator.samples // batch\_size,

epochs=30,

validation\_data=validation\_generator,

validation\_steps=validation\_generator.samples // batch\_size

)

# Plot training & validation accuracy values

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')

plt.show()

model.save('/content/drive/MyDrive/waste project/waste\_classification\_mobilenetv2.pb')

# Load your trained model

model = tf.keras.models.load\_model('/content/drive/MyDrive/waste project/waste\_classification\_mobilenetv2.h5')

# Predict classes using the model

predictions = model.predict(test\_generator)

predicted\_classes = np.argmax(predictions, axis=1)

# Get true labels

true\_classes = test\_generator.classes

# Create the confusion matrix

conf\_matrix = confusion\_matrix(true\_classes, predicted\_classes)

# Print classification report

class\_names = ['Plastic', 'Glass', 'Metal', 'Paper', 'Cardboard', 'trash']

print("\nClassification Report:")

print(classification\_report(true\_classes, predicted\_classes, target\_names=class\_names))

import matplotlib.pyplot as plt

import seaborn as sns

# Your confusion matrix (conf\_matrix) from previous code

# Ensure it's computed before this point

# Define class names

class\_names = ['Plastic', 'Glass', 'Metal', 'Paper', 'Cardboard', 'trash']

# Create a heatmap for the confusion matrix

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

sns.heatmap(conf\_matrix, annot=True, cmap='Blues', fmt='d', xticklabels=class\_names, yticklabels=class\_names)

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()

import cv2

from google.colab.patches import cv2\_imshow

# Function to preprocess the image

def preprocess\_image(image\_path):

img = cv2.imread(image\_path)

img = cv2.resize(img, (224, 224))

img = img / 255.0 # Normalize pixel values

img = np.expand\_dims(img, axis=0) # Add batch dimension

return img

# Function to predict and visualize bounding boxes

def predict\_and\_visualize(image\_path):

img = preprocess\_image(image\_path)

predictions = model.predict(img)

# Define class labels (replace with your own class labels)

class\_labels = ['plastic', 'glass', 'metal', 'paper', 'cardboard', 'trash']

# Get the predicted class and confidence level

predicted\_class = np.argmax(predictions)

confidence = np.max(predictions) \* 100

# Read the image using OpenCV for drawing bounding boxes

image = cv2.imread(image\_path)

image = cv2.resize(image, (224, 224))

# Draw bounding box on the image

cv2.putText(image, f'{class\_labels[predicted\_class]} - {confidence:.2f}%', (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2)

cv2.rectangle(image, (0, 0), (image.shape[1], image.shape[0]), (0, 255, 0), 3)

# Show the image with bounding box and classification

cv2\_imshow(image)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Test the model by passing an image path

image\_path\_to\_test = '/content/drive/MyDrive/waste project/data/valid/plastic/plastic121.jpg'

predict\_and\_visualize(image\_path\_to\_test)