**FOR SPLITTING THE DATASET INTO TRAIN, TEST, VAL FOLDERS.**

! pip install python\_splitter

import python\_splitter

python\_splitter.split\_from\_folder("C:/Users/yashs/SmartRecyclingSystem/TrashBox-main/TrashBox-main/TrashBox\_train\_set", train=0.7, test=0.15, val=0.15)

**IMPORTING ALL THE NECESSARY PACKAGES.**

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import MobileNetV2

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

import matplotlib.pyplot as plt

import os

**ASSIGNING DIRECTORY LOCATIONS TO RESPECTIVE VARIABLES.**

train\_dir = "C:/Users/yashs/SmartRecyclingSystem/Train\_Test\_Folder/train"

validation\_dir = "C:/Users/yashs/SmartRecyclingSystem/Train\_Test\_Folder/val"

test\_dir = "C:/Users/yashs/SmartRecyclingSystem/Train\_Test\_Folder/test"

# Data augmentation for training dataset

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'

)

# Validation and Test dataset generators (without augmentation)

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

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

# Load images from directories

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

train\_dir,

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical'

)

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

validation\_dir,

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical'

)

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

test\_dir,

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical'

)

Found 9994 images belonging to 7 classes.

Found 2140 images belonging to 7 classes.

Found 2145 images belonging to 7 classes.

**LOAD PRE-TRAINED MOBILENETV2**

# Load pre-trained MobileNetV2 without the top layers

base\_model = MobileNetV2(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

# Freeze the base model layers to prevent them from being trained

for layer in base\_model.layers:

layer.trainable = False

# Add custom layers on top of the pre-trained base model

x = base\_model.output

x = GlobalAveragePooling2D()(x) # Global average pooling reduces the tensor size

x = Dense(128, activation='relu')(x) # Add a fully connected layer

predictions = Dense(train\_generator.num\_classes, activation='softmax')(x) # Output layer with softmax

# Build the model

model = Model(inputs=base\_model.input, outputs=predictions)

# Compile the model

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

# Show the model summary

model.summary()

**Total params:** 2,422,855 (9.24 MB)

**Trainable params:** 164,871 (644.03 KB)

**Non-trainable params:** 2,257,984 (8.61 MB)

**VALIDATING IMAGES IN DATASET.**

from PIL import Image

import os

# Define your dataset directories

train\_dir = "C:/Users/yashs/SmartRecyclingSystem/Train\_Test\_Folder/train"

validation\_dir = "C:/Users/yashs/SmartRecyclingSystem/Train\_Test\_Folder/val"

def validate\_images(directory):

for subdir, \_, files in os.walk(directory):

for file in files:

try:

img\_path = os.path.join(subdir, file)

img = Image.open(img\_path)

img.verify() # Verify if the image is valid

except (IOError, SyntaxError) as e:

print(f"Corrupted image: {img\_path}")

# Validate images in the specified directories

validate\_images(train\_dir)

validate\_images(validation\_dir)

**TRAINING THE MODEL.**

# Train the model

history = model.fit(

train\_generator,

epochs=10, # You can adjust the number of epochs

validation\_data=validation\_generator

)

Epoch 1/10

**313/313** ━━━━━━━━━━━━━━━━━━━━ **453s** 1s/step - accuracy: 0.6304 - loss: 1.0584 - val\_accuracy: 0.7780 - val\_loss: 0.6346

Epoch 2/10

**313/313** ━━━━━━━━━━━━━━━━━━━━ **417s** 1s/step - accuracy: 0.7897 - loss: 0.6207 - val\_accuracy: 0.7991 - val\_loss: 0.5909

Epoch 3/10

**313/313** ━━━━━━━━━━━━━━━━━━━━ **429s** 1s/step - accuracy: 0.8180 - loss: 0.5308 - val\_accuracy: 0.8033 - val\_loss: 0.5757

Epoch 4/10

**313/313** ━━━━━━━━━━━━━━━━━━━━ **419s** 1s/step - accuracy: 0.8257 - loss: 0.4978 - val\_accuracy: 0.8206 - val\_loss: 0.5471

Epoch 5/10

**313/313** ━━━━━━━━━━━━━━━━━━━━ **417s** 1s/step - accuracy: 0.8515 - loss: 0.4341 - val\_accuracy: 0.8187 - val\_loss: 0.5454

Epoch 6/10

**313/313** ━━━━━━━━━━━━━━━━━━━━ **414s** 1s/step - accuracy: 0.8628 - loss: 0.3968 - val\_accuracy: 0.8182 - val\_loss: 0.5362

Epoch 7/10

**313/313** ━━━━━━━━━━━━━━━━━━━━ **751s** 2s/step - accuracy: 0.8688 - loss: 0.3683 - val\_accuracy: 0.8192 - val\_loss: 0.5319

Epoch 8/10

**313/313** ━━━━━━━━━━━━━━━━━━━━ **271s** 867ms/step - accuracy: 0.8893 - loss: 0.3252 - val\_accuracy: 0.8136 - val\_loss: 0.5772

Epoch 9/10

**313/313** ━━━━━━━━━━━━━━━━━━━━ **268s** 857ms/step - accuracy: 0.8907 - loss: 0.3155 - val\_accuracy: 0.8140 - val\_loss: 0.5724

Epoch 10/10

**313/313** ━━━━━━━━━━━━━━━━━━━━ **258s** 825ms/step - accuracy: 0.8972 - loss: 0.2944 - val\_accuracy: 0.8276 - val\_loss: 0.5467

**PLOT TRAINING AND VALIDATION ACCURACY VALUES.**

# Plot training & validation accuracy values

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

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

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

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

plt.show()

# Plot training & validation loss values

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

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

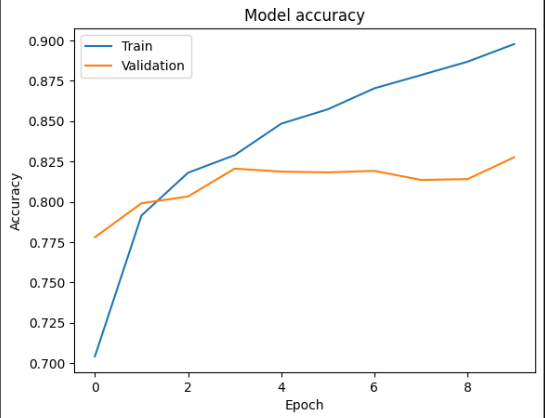
plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

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

plt.show()





**FINE-TUNING A PRE-TRAINED MODEL TO IMPROVE ITS PERFORMANCE ON NEW DATASET.**

# Unfreeze some of the layers in the base model

for layer in base\_model.layers[-20:]:

layer.trainable = True

# Recompile the model with a lower learning rate

model.compile(optimizer=Adam(learning\_rate=1e-5), loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model again with fine-tuning

history\_fine = model.fit(

train\_generator,

epochs=5,

validation\_data=validation\_generator

)

Epoch 1/5

**313/313** ━━━━━━━━━━━━━━━━━━━━ **283s** 887ms/step - accuracy: 0.7907 - loss: 0.6086 - val\_accuracy: 0.8341 - val\_loss: 0.5487

Epoch 2/5

**313/313** ━━━━━━━━━━━━━━━━━━━━ **275s** 878ms/step - accuracy: 0.8496 - loss: 0.4150 - val\_accuracy: 0.8407 - val\_loss: 0.5427

Epoch 3/5

**313/313** ━━━━━━━━━━━━━━━━━━━━ **1516s** 5s/step - accuracy: 0.8709 - loss: 0.3601 - val\_accuracy: 0.8402 - val\_loss: 0.5363

Epoch 4/5

**313/313** ━━━━━━━━━━━━━━━━━━━━ **276s** 883ms/step - accuracy: 0.8822 - loss: 0.3425 - val\_accuracy: 0.8430 - val\_loss: 0.5270

Epoch 5/5

**313/313** ━━━━━━━━━━━━━━━━━━━━ **274s** 875ms/step - accuracy: 0.8848 - loss: 0.3208 - val\_accuracy: 0.8439 - val\_loss: 0.5146

**VALIDATING TEST IMAGE DATASET.**

validate\_images(test\_dir)

**EVALUATING ON TEST SET.**

# Evaluate on the test set

test\_loss, test\_acc = model.evaluate(test\_generator)

print(f"Test accuracy: {test\_acc \* 100:.2f}%")

**68/68** ━━━━━━━━━━━━━━━━━━━━ **35s** 517ms/step - accuracy: 0.8349 - loss: 0.4944

**Test accuracy: 84.94%**

**SAVING THE TRAINED MODEL FOR LATER USE.**

**# Save the trained model**

**model.save('smart\_recycling\_model.keras')**

**FOR LOADING AN IMAGE FOR PREDICTION.**

import numpy as np

from tensorflow.keras.preprocessing import image

# Load an image for prediction

img\_path = "C:/Users/yashs/Downloads/download.jpg"

img = image.load\_img(img\_path, target\_size=(224, 224))

img\_array = image.img\_to\_array(img)

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

img\_array /= 255. # Normalize the image

# Predict the class of the waste item

predictions = model.predict(img\_array)

predicted\_class = np.argmax(predictions[0])

class\_labels = list(train\_generator.class\_indices.keys()) # Get class labels

print(f'This waste item is: {class\_labels[predicted\_class]}')