CNN FROM SCRATCH:

```
import tensorflow as tf
import keras
from keras._tf_keras.keras.preprocessing.image import ImageDataGenerator

train_datagen1=
ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
!unzip -o "/content/drive/MyDrive/dataset_new.zip" -d "/usr/local"
training_set1=train_datagen1.flow_from_directory('/usr/local/dataset_new/plastic_ewaste/train',
target_size=(128,128),batch_size=32,class_mode='binary')
test_datagen1=ImageDataGenerator(rescale=1./255)
test_set1=test_datagen1.flow_from_directory('/usr/local/dataset_new/plastic_ewaste/test',
target_size=(128,128),batch_size=32,class_mode='binary')
val_datagen1=ImageDataGenerator(rescale=1./255)
val_set1=val_datagen1.flow_from_directory('/usr/local/dataset_new/plastic_ewaste/val',
target_size=(128,128),batch_size=32,class_mode='binary')

cnn1 = tf.keras.models.Sequential([
   tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(128,128,3)),
   tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

   tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
   tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

   tf.keras.layers.Flatten(),
   tf.keras.layers.Dense(128, activation='relu'),
   tf.keras.layers.Dropout(0.5),   # 🔹 Dropout layer added here
   tf.keras.layers.Dense(1, activation='sigmoid')
])

cnn1.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
import numpy as np
from collections import Counter

counter = Counter(training_set1.classes)  # classes → 0=E-WASTE, 1=PLASTIC WASTE
print(counter)

# Example output: Counter({1: 800, 0: 500})
total = float(sum(counter.values()))
class_weight = {
   0: total / (2 * counter[0]),  # weight for E-WASTE
   1: total / (2 * counter[1])   # weight for PLASTIC
}
print("Class Weights:", class_weight)
```

```
history = cnn1.fit(
    x=training_set1,
    validation_data=val_set1,
    epochs=50,
    callbacks=[early_stop],   # 🔹 Added callback
    class_weight=class_weight
)
from google.colab import output
from IPython.display import display, Javascript
from google.colab.output import eval_js
import cv2
import numpy as np
from PIL import Image
import io
import base64

def take_photo(filename='photo.jpg', quality=0.8):
    js = Javascript('''
    async function takePhoto(quality) {
      const div = document.createElement('div');
      const capture = document.createElement('button');
      capture.textContent = 'Capture';
      div.appendChild(capture);

      const video = document.createElement('video');
      video.style.display = 'block';
      const stream = await navigator.mediaDevices.getUserMedia({video: true});

      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      await video.play();

      // Resize the output to fit the video element.
      google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);

      // Wait for Capture to be clicked.
      await new Promise((resolve) => capture.onclick = resolve);

      const canvas = document.createElement('canvas');
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      canvas.getContext('2d').drawImage(video, 0, 0);
      stream.getTracks().forEach(track => track.stop());
      div.remove();
      return canvas.toDataURL('image/jpeg', quality);
    }
    ''')
    display(js)
    data = eval_js('takePhoto({})'.format(quality))
    binary = base64.b64decode(data.split(',')[1])
    with open(filename, 'wb') as f:
        f.write(binary)
    return filename

# Capture photo via webcam
photo_filename = take_photo()
print('Saved to {}'.format(photo_filename))
from keras.preprocessing import image

img = image.load_img(photo_filename, target_size=(128, 128))
img_array = image.img_to_array(img) / 255.0
img_array = np.expand_dims(img_array, axis=0)
prediction = cnn1.predict(img_array)

# Interpret prediction
prediction_label = 'PLASTIC WASTE' if prediction[0][0] >= 0.5 else 'E WASTE'
print(f'Prediction: {prediction_label}')
cnn1.save('model.h5')
print("Model saved as model.h5")
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import numpy as np

# Get true labels and predictions
y_true = []
y_pred = []
for batch in test_set1:
    X, y = batch
    preds = cnn1.predict(X)
    y_true.extend(y.flatten())
    y_pred.extend((preds > 0.5).astype(int).flatten())
    if len(y_true) >= test_set1.samples:
        break

cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['E WASTE', 'PLASTIC WASTE'])
disp.plot()
from sklearn.metrics import classification_report

# Generate classification report
report = classification_report(y_true, y_pred, target_names=['E WASTE', 'PLASTIC WASTE'])
print(report)
train_acc = history.history['accuracy'][-1]
val_acc = history.history['val_accuracy'][-1]

# Evaluate on test set to get test accuracy
```

```
loss, test_acc = cnn1.evaluate(test_set1)
print(f"Training Accuracy: {train_acc}")
print(f"Validation Accuracy: {val_acc}")
print(f"Test Accuracy: {test_acc}")
  import matplotlib.pyplot as plt

  acc = history.history['accuracy']
  val_acc = history.history['val_accuracy']
  loss = history.history['loss']
  val_loss = history.history['val_loss']
  epochs_range = range(len(acc))

  plt.figure(figsize=(12, 5))
  plt.subplot(1, 2, 1)
  plt.plot(epochs_range, acc, label='Training Accuracy')
  plt.plot(epochs_range, val_acc, label='Validation Accuracy')
  plt.legend(loc='lower right')
  plt.title('Training and Validation Accuracy')

  plt.subplot(1, 2, 2)
  plt.plot(epochs_range, loss, label='Training Loss')
  plt.plot(epochs_range, val_loss, label='Validation Loss')
  plt.legend(loc='upper right')
  plt.title('Training and Validation Loss')
  plt.show()
cnn1.save('/content/drive/MyDrive/cnn_scratch.h5')
```

NASNET MOBILE MODEL :

```
import tensorflow as tf
from keras._tf_keras.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.applications import NASNetMobile
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter

# ====================
# DATASET LOADING
# ====================
train_datagen = ImageDataGenerator(rescale=1./255,
                      shear_range=0.2,
                      zoom_range=0.2,
                      horizontal_flip=True)
!unzip -o "/content/drive/MyDrive/dataset_new.zip" -d "/usr/local"
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_set = train_datagen.flow_from_directory(
    '/usr/local/dataset_new/plastic_ewaste/train',
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary')

val_set = val_datagen.flow_from_directory(
    '/usr/local/dataset_new/plastic_ewaste/val',
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary')

test_set = test_datagen.flow_from_directory(
    '/usr/local/dataset_new/plastic_ewaste/test',
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary')

# Handle class imbalance
counter = Counter(train_set.classes)
total = float(sum(counter.values()))
class_weight = {
    0: total / (2 * counter[0]),
    1: total / (2 * counter[1])
}
print("Class Weights:", class_weight)

# Early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# ====================
# NASNetMobile MODEL
# ====================
base_model = NASNetMobile(input_shape=(128,128,3),
                          include_top=False,
                          weights='imagenet')

base_model.trainable = False  # Freeze convolutional base

model_nas = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model_nas.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
# =====================
# TRAINING
# =====================
history_nas = model_nas.fit(
   train_set,
   validation_data=val_set,
   epochs=30,
   callbacks=[early_stop],
   class_weight=class_weight
)

# =====================
# EVALUATION
# =====================
loss, acc = model_nas.evaluate(test_set)
print(f"NASNetMobile Test Accuracy: {acc:.4f}")

# Training & validation accuracy
train_acc_nas = history_nas.history['accuracy'][-1]
val_acc_nas = history_nas.history['val_accuracy'][-1]

print(f"NASNetMobile Training Accuracy: {train_acc_nas:.4f}")
print(f"NASNetMobile Validation Accuracy: {val_acc_nas:.4f}")

# =====================
# CLASSIFICATION REPORT
# =====================
y_true, y_pred = [], []
for batch in test_set:
   X, y = batch
   preds = model_nas.predict(X)
   y_true.extend(y.flatten())
   y_pred.extend((preds > 0.5).astype(int).flatten())
   if len(y_true) >= test_set.samples:
      break

print("=== NASNetMobile Performance ===")
print(classification_report(y_true, y_pred, target_names=['E-WASTE','PLASTIC WASTE']))

cm = confusion_matrix(y_true, y_pred)
ConfusionMatrixDisplay(cm, display_labels=['E-WASTE','PLASTIC WASTE']).plot()

# =====================
# SAVE MODEL
# =====================
model_nas.save("/content/drive/MyDrive/nasnetmobile_model.h5")
model_nas.save("/content/drive/MyDrive/nasnetmobile_model.keras")
print("✅ NASNetMobile model saved as nasnetmobile_model.h5 and as .keras")
```

```
import matplotlib.pyplot as plt

# =========================
# Plot Accuracy and Loss
# =========================
def plot_training_history(history, model_name="Model"):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)

    # ---- Accuracy Plot ----
    plt.figure(figsize=(8,5))
    plt.plot(epochs, acc, 'b-', label='Training Accuracy')
    plt.plot(epochs, val_acc, 'orange', label='Validation Accuracy')
    plt.title(f'{model_name} - Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)
    plt.show()

    # ---- Loss Plot ----
    plt.figure(figsize=(8,5))
    plt.plot(epochs, loss, 'b-', label='Training Loss')
    plt.plot(epochs, val_loss, 'orange', label='Validation Loss')
    plt.title(f'{model_name} - Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.show()

# =========================
# Call function for NASNet
# =========================
plot_training_history(history_nas, model_name="NASNetMobile")
```

INCEPTIONV3 MODEL

```
import tensorflow as tf
from keras._tf_keras.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.applications import InceptionV3
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter
```

```
# ====================
# DATASET LOADING
# ====================
train_datagen = ImageDataGenerator(
   rescale=1./255,
   shear_range=0.2,
   zoom_range=0.2,
   horizontal_flip=True
)

!unzip -o "/content/drive/MyDrive/dataset_new.zip" -d "/usr/local"

val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_set = train_datagen.flow_from_directory(
   '/usr/local/dataset_new/plastic_ewaste/train',
   target_size=(150, 150),   # Inception default is 299x299; 150x150 works for faster training
   batch_size=32,
   class_mode='binary'
)

val_set = val_datagen.flow_from_directory(
   '/usr/local/dataset_new/plastic_ewaste/val',
   target_size=(150, 150),
   batch_size=32,
   class_mode='binary'
)

test_set = test_datagen.flow_from_directory(
   '/usr/local/dataset_new/plastic_ewaste/test',
   target_size=(150, 150),
   batch_size=32,
   class_mode='binary'
)

# ====================
# HANDLE CLASS IMBALANCE
# ====================
from collections import Counter
counter = Counter(train_set.classes)
total = float(sum(counter.values()))
class_weight = {
   0: total / (2 * counter[0]),  # E-waste
   1: total / (2 * counter[1])   # Plastic
}
print("Class Weights:", class_weight)
```

```
# ====================
# MODEL CREATION
# ====================
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

base_model = InceptionV3(
    input_shape=(150, 150, 3),
    include_top=False,
    weights='imagenet'
)
base_model.trainable = False  # Freeze convolutional layers

# Build model
from tensorflow.keras import Input, Model
inputs = Input(shape=(150,150,3))
x = base_model(inputs, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dense(128, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(x)
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)

inception_model = Model(inputs, outputs)

inception_model.compile(optimizer='adam',
                        loss='binary_crossentropy',
                        metrics=['accuracy'])

# ====================
# TRAINING
# ====================
history_incep = inception_model.fit(
    train_set,
    validation_data=val_set,
    epochs=30,
    callbacks=[early_stop],
    class_weight=class_weight
)

# ====================
# EVALUATION
# ====================
loss, acc = inception_model.evaluate(test_set)
print(f"InceptionV3 Test Accuracy: {acc:.4f}")

train_acc = history_incep.history['accuracy'][-1]
val_acc = history_incep.history['val_accuracy'][-1]
print(f"InceptionV3 Training Accuracy: {train_acc:.4f}")
print(f"InceptionV3 Validation Accuracy: {val_acc:.4f}")
```

```
# =====================
# CLASSIFICATION REPORT
# =====================
y_true, y_pred = [], []
for batch in test_set:
    X, y = batch
    preds = inception_model.predict(X)
    y_true.extend(y.flatten())
    y_pred.extend((preds > 0.5).astype(int).flatten())
    if len(y_true) >= test_set.samples:
        break

print("=== InceptionV3 Performance ===")
print(classification_report(y_true, y_pred, target_names=['E-WASTE','PLASTIC WASTE']))

cm = confusion_matrix(y_true, y_pred)
ConfusionMatrixDisplay(cm, display_labels=['E-WASTE','PLASTIC WASTE']).plot()

# =====================
# SAVE MODELS
# =====================
inception_model.save("/content/drive/MyDrive/inceptionv3_model.h5")
inception_model.save("/content/drive/MyDrive/inceptionv3_model.keras")
print("✅ InceptionV3 model saved as both .h5 and .keras formats!")

# =====================
# ACCURACY & LOSS PLOTS
# =====================
def plot_training_history(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)

    # ---- Accuracy Plot ----
    plt.figure(figsize=(8,5))
    plt.plot(epochs, acc, 'b-', label='Training Accuracy')
    plt.plot(epochs, val_acc, 'orange', label='Validation Accuracy')
    plt.title('InceptionV3 - Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)
    plt.show()

    # ---- Loss Plot ----
    plt.figure(figsize=(8,5))
    plt.plot(epochs, loss, 'b-', label='Training Loss')
    plt.plot(epochs, val_loss, 'orange', label='Validation Loss')
    plt.title('InceptionV3 - Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.show()

plot_training_history(history_incep)
```

MOBILENETV2 MODEL

```
import tensorflow as tf
from keras._tf_keras.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter

# ====================
# DATASET LOADING
# ====================
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)
!unzip -o "/content/drive/MyDrive/dataset_new.zip" -d "/usr/local"
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_set = train_datagen.flow_from_directory(
    '/usr/local/dataset_new/plastic_ewaste/train',
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary')

val_set = val_datagen.flow_from_directory(
    '/usr/local/dataset_new/plastic_ewaste/val',
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary')

test_set = test_datagen.flow_from_directory(
    '/usr/local/dataset_new/plastic_ewaste/test',
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary')
```

```
# Handle class imbalance
counter = Counter(train_set.classes)
total = float(sum(counter.values()))
class_weight = {
   0: total / (2 * counter[0]),
   1: total / (2 * counter[1])
}
print("Class Weights:", class_weight)

# Early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
# ====================
# MODEL 3: MobileNetV2 Transfer Learning
# ====================
from tensorflow.keras import Input, Model
from tensorflow.keras.applications import MobileNetV2

# Load base model
base_model = MobileNetV2(input_shape=(128,128,3),
                         include_top=False,
                         weights='imagenet')
base_model.trainable = False  # freeze pretrained layers

# Functional API model
inputs = Input(shape=(128,128,3))
x = base_model(inputs, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dense(128, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(x)
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)

mobilenet_model = Model(inputs, outputs)

mobilenet_model.compile(optimizer='adam',
                        loss='binary_crossentropy',
                        metrics=['accuracy'])

# Train
history3 = mobilenet_model.fit(
   train_set,
   validation_data=val_set,
   epochs=30,
   callbacks=[early_stop],
   class_weight=class_weight
)

# Save (use .keras format!)
# Evaluate
loss, acc = mobilenet_model.evaluate(test_set)
print(f"MobileNetV2 Test Accuracy: {acc:.4f}")

# Classification report
y_true, y_pred = [], []
for batch in test_set:
    X, y = batch
    preds = mobilenet_model.predict(X)
    y_true.extend(y.flatten())
    y_pred.extend((preds > 0.5).astype(int).flatten())
    if len(y_true) >= test_set.samples:
        break

print(classification_report(y_true, y_pred, target_names=['E-WASTE','PLASTIC']))
cm = confusion_matrix(y_true, y_pred)
ConfusionMatrixDisplay(cm, display_labels=['E-WASTE','PLASTIC']).plot()
from google.colab import files
from keras.preprocessing import image

# Upload image
uploaded = files.upload()

for fn in uploaded.keys():
    img_path = fn
    img = image.load_img(img_path, target_size=(128,128))
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    # Change model1 → model2 or model3 depending on which one you trained
    prediction = mobilenet_model.predict(img_array)

    # Interpret prediction
    label = "PLASTIC WASTE" if prediction[0][0] >= 0.5 else "E-WASTE"
    print(f"Prediction for {fn}: {label} (score={prediction[0][0]:.4f})")
from IPython.display import display, Javascript
from google.colab.output import eval_js
import cv2, io, base64
import numpy as np
from PIL import Image

def take_photo(filename='photo.jpg', quality=0.8):
    js = Javascript('''
      async function takePhoto(quality) {
        const div = document.createElement('div');
        const capture = document.createElement('button');
        capture.textContent = 'Capture';
        div.appendChild(capture);
        const video = document.createElement('video');
        video.style.display = 'block';
        const stream = await navigator.mediaDevices.getUserMedia({video: true});

        document.body.appendChild(div);
        div.appendChild(video);
        video.srcObject = stream;
        await video.play();

        google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);

        await new Promise((resolve) => capture.onclick = resolve);

        const canvas = document.createElement('canvas');
        canvas.width = video.videoWidth;
        canvas.height = video.videoHeight;
        canvas.getContext('2d').drawImage(video, 0, 0);
        stream.getTracks().forEach(track => track.stop());
        div.remove();
        return canvas.toDataURL('image/jpeg', quality);
      }
    ''')
    display(js)
    data = eval_js('takePhoto({})'.format(quality))
    binary = base64.b64decode(data.split(',')[1])
    with open(filename, 'wb') as f:
        f.write(binary)
    return filename

# Capture photo
photo_filename = take_photo()
print("Saved image:", photo_filename)

# Preprocess and predict
img = image.load_img(photo_filename, target_size=(128,128))
img_array = image.img_to_array(img) / 255.0
img_array = np.expand_dims(img_array, axis=0)

prediction = mobilenet_model.predict(img_array)  # change model2 → model1 or model3
label = "PLASTIC WASTE" if prediction[0][0] >= 0.5 else "E-WASTE"
print(f"Prediction: {label} (score={prediction[0][0]:.4f})")
mobilenet_model.save("/content/drive/MyDrive/mobilenet.keras")
mobilenet_model.save("/content/drive/MyDrive/mobilenet.h5")
import matplotlib.pyplot as plt

# ========================
# Plot Accuracy and Loss
# =========================
def plot_training_history(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)

    # ---- Accuracy Plot ----
    plt.figure(figsize=(8,5))
    plt.plot(epochs, acc, 'b-', label='Training Accuracy')
    plt.plot(epochs, val_acc, 'orange', label='Validation Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)
    plt.show()

    # ---- Loss Plot ----
    plt.figure(figsize=(8,5))
    plt.plot(epochs, loss, 'b-', label='Training Loss')
    plt.plot(epochs, val_loss, 'orange', label='Validation Loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.show()

# Call function with your history object
plot_training_history(history3)
```

RESNET50 MODEL:

```
import tensorflow as tf
from keras._tf_keras.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter

# ====================
# DATASET LOADING
# ====================
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)
```

```
!unzip -o "/content/drive/MyDrive/dataset_new.zip" -d "/usr/local"

val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_set = train_datagen.flow_from_directory(
    '/usr/local/dataset_new/plastic_ewaste/train',
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary')

val_set = val_datagen.flow_from_directory(
    '/usr/local/dataset_new/plastic_ewaste/val',
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary')

test_set = test_datagen.flow_from_directory(
    '/usr/local/dataset_new/plastic_ewaste/test',
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary')

# Handle class imbalance
counter = Counter(train_set.classes)
total = float(sum(counter.values()))
class_weight = {
    0: total / (2 * counter[0]),
    1: total / (2 * counter[1])
}
print("Class Weights:", class_weight)

# Early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# ====================
# RESNET50 MODEL
# ====================
base_model = tf.keras.applications.ResNet50(
    input_shape=(128,128,3),
    include_top=False,
    weights='imagenet'
)

base_model.trainable = False  # freeze pretrained layers

resnet_model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

resnet_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# ====================
# TRAINING
# ====================
history_resnet = resnet_model.fit(
    train_set,
    validation_data=val_set,
    epochs=30,
    callbacks=[early_stop],
    class_weight=class_weight
)

# ====================
# EVALUATION
# ====================
loss, acc = resnet_model.evaluate(test_set)
print(f"ResNet50 Test Accuracy: {acc:.4f}")

# Training & validation accuracy
train_acc_resnet = history_resnet.history['accuracy'][-1]
val_acc_resnet = history_resnet.history['val_accuracy'][-1]
print(f"Training Accuracy: {train_acc_resnet:.4f}")
print(f"Validation Accuracy: {val_acc_resnet:.4f}")

# Classification report
y_true, y_pred = [], []
for batch in test_set:
    X, y = batch
    preds = resnet_model.predict(X)
    y_true.extend(y.flatten())
    y_pred.extend((preds > 0.5).astype(int).flatten())
    if len(y_true) >= test_set.samples:
        break

print("=== ResNet50 Performance ===")
print(classification_report(y_true, y_pred, target_names=['E-WASTE','PLASTIC WASTE']))

cm = confusion_matrix(y_true, y_pred)
ConfusionMatrixDisplay(cm, display_labels=['E-WASTE','PLASTIC WASTE']).plot()
```

```
# =====================
# SAVE MODEL
# =====================
resnet_model.save("/content/drive/MyDrive/resnet50_model.h5")
print("✅ ResNet50 model saved as resnet50_model.h5")

# =====================
# PLOT TRAINING HISTORY
# =====================
def plot_training_history(history, model_name="Model"):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)

    plt.figure(figsize=(8,5))
    plt.plot(epochs, acc, 'b-', label='Training Accuracy')
    plt.plot(epochs, val_acc, 'orange', label='Validation Accuracy')
    plt.title(f'{model_name} - Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)
    plt.show()

    plt.figure(figsize=(8,5))
    plt.plot(epochs, loss, 'b-', label='Training Loss')
    plt.plot(epochs, val_loss, 'orange', label='Validation Loss')
    plt.title(f'{model_name} - Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.show()

plot_training_history(history_resnet, "ResNet50")
resnet_model.save("/content/drive/MyDrive/resnet50_model.keras")
print("✅ Model also saved as resnet50_model.keras")
```