Import necessarry library dependencies

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.applications import EfficientNetV2B0
from tensorflow.keras.applications.efficientnet import preprocess_input
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

Import datasets

```
#Dataset paths
trainpath = r'/content/drive/MyDrive/modified-dataset/train'
validpath = r'/content/drive/MyDrive/modified-dataset/val'
testpath  = r'/content/drive/MyDrive/modified-dataset/test'
```

Understand the Data

```
# 1EXPLORE AND UNDERSTAND THE DATA
IMG_SIZE = (128, 128)
BATCH_SIZE = 32

datatrain = tf.keras.utils.image_dataset_from_directory(trainpath, shuffle=True, image_size=IMG_SIZE, batch_size=BATCH_SIZE)
datavalid = tf.keras.utils.image_dataset_from_directory(validpath, shuffle=True, image_size=IMG_SIZE, batch_size=BATCH_SIZE)
datatest  = tf.keras.utils.image_dataset_from_directory(testpath, shuffle=False, image_size=IMG_SIZE, batch_size=BATCH_SIZE)

class_names = datatrain.class_names
print(f"Classes: {class_names}")
```

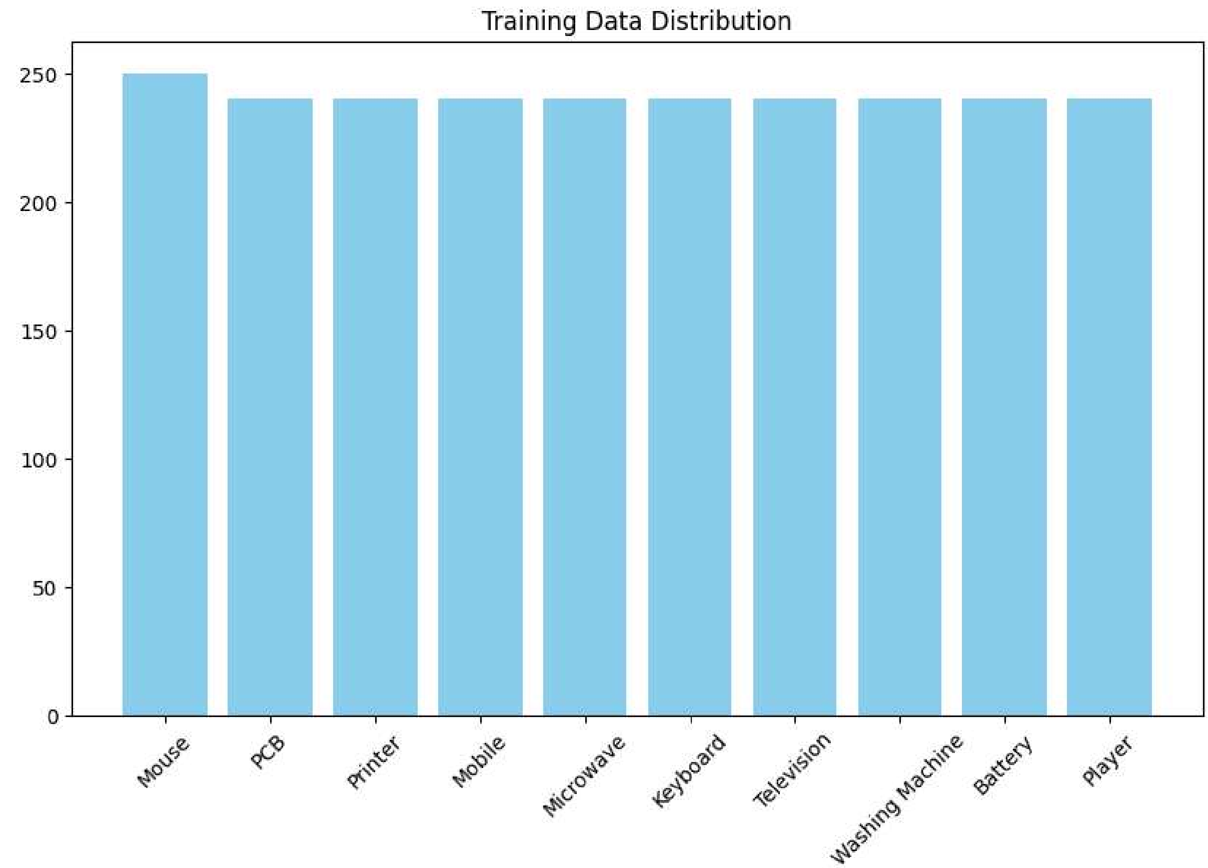
```
Found 2410 files belonging to 10 classes.
Found 300 files belonging to 10 classes.
Found 310 files belonging to 10 classes.
Classes: ['Battery', 'Keyboard', 'Microwave', 'Mobile', 'Mouse', 'PCB', 'Player', 'Printer', 'Television', 'Washing Machine'
```

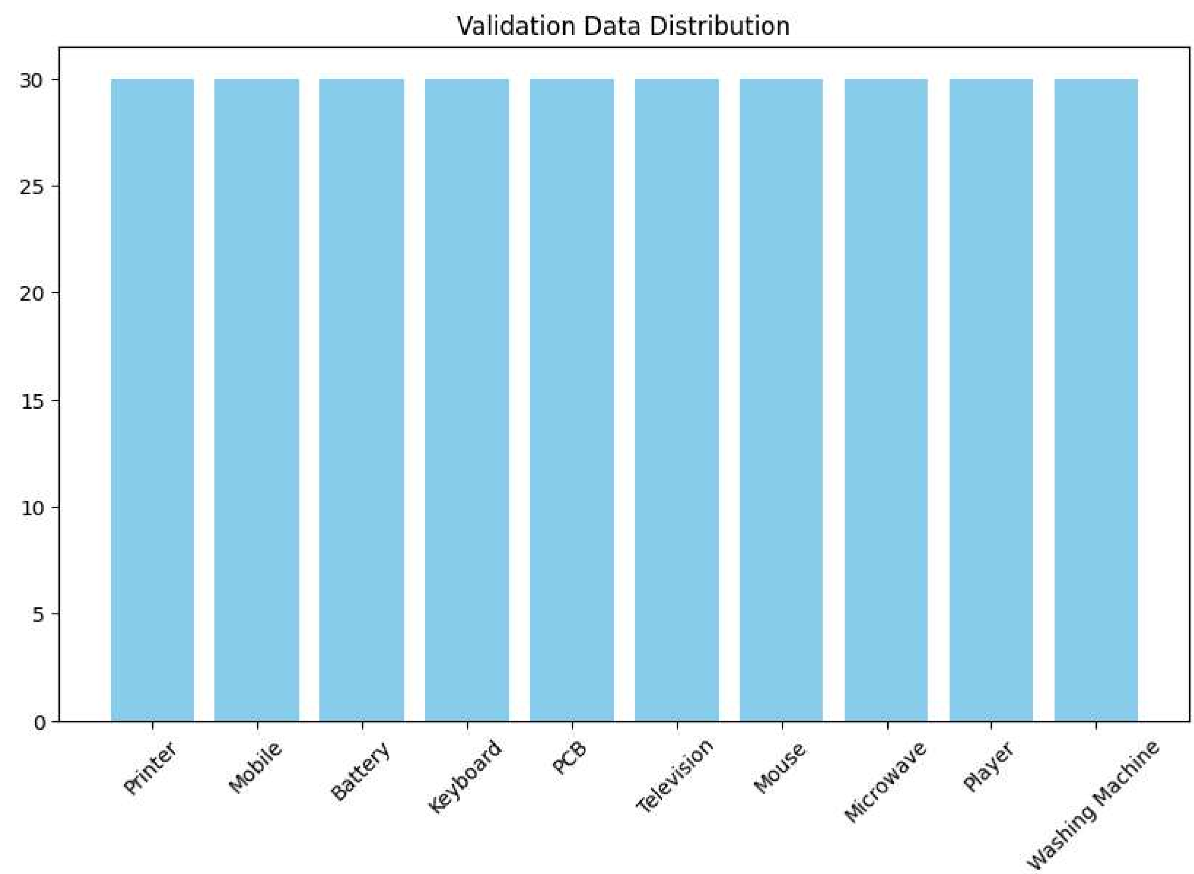
Visualize samples

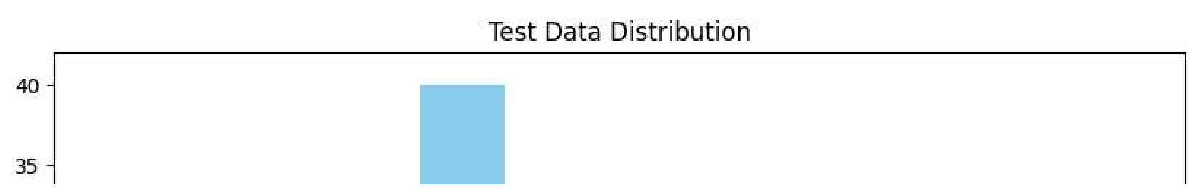
```
# Visualize samples
plt.figure(figsize=(10,10))
for images, labels in datatrain.take(1):
    for i in range(9):
        ax = plt.subplot(3,3,i+1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
plt.show()
```

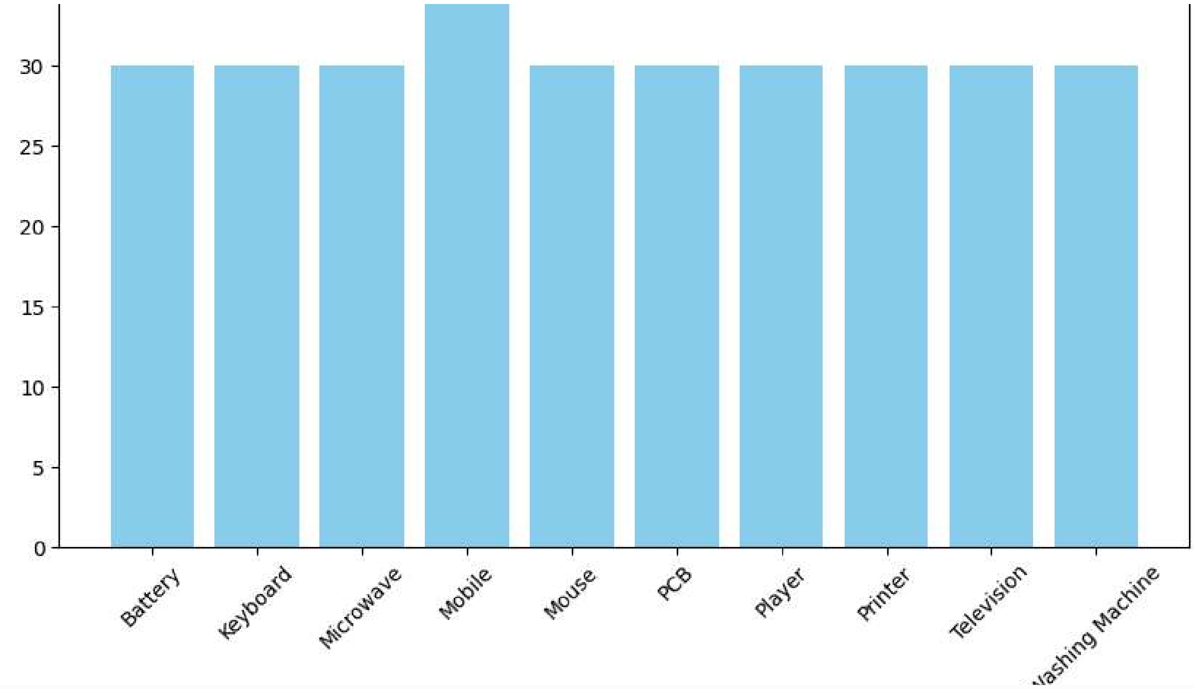
Plot class distribution

```
# Plot class distribution
def plot_class_distribution(dataset, title):
    counts = {}
    for _, labels in dataset:
        for label in labels.numpy():
            class_name = dataset.class_names[label]
            counts[class_name] = counts.get(class_name, 0) + 1
    plt.figure(figsize=(10,6))
    plt.bar(counts.keys(), counts.values(), color='skyblue')
    plt.title(title)
    plt.xticks(rotation=45)
    plt.show()

plot_class_distribution(datatrain, "Training Data Distribution")
plot_class_distribution(datavalid, "Validation Data Distribution")
plot_class_distribution(datatest, "Test Data Distribution")
```


Training Data Distribution
250
200
150
100
50
0
Mouse
PCB
Printer
Mobile
Microwave
Keyboard
Television
Washing Machine
Battery
Player


Validation Data Distribution
30
25
20
15
10
5
0
Printer
Mobile
Battery
Keyboard
PCB
Television
Mouse
Microwave
Player
Washing Machine


Test Data Distribution
40
35



DATA PREPROCESSING

```
# DATA PREPROCESSING / PREPARATION
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
])

datatrain = datatrain.prefetch(tf.data.AUTOTUNE)
datavalid = datavalid.prefetch(tf.data.AUTOTUNE)
datatest  = datatest.prefetch(tf.data.AUTOTUNE)
```

MODEL SELECTION USING EFFICIENTNET

```
# MODEL SELECTION
base_model = EfficientNetV2B0(input_shape=IMG_SIZE+(3,), include_top=False, weights='imagenet')
for layer in base_model.layers[:100]:
    layer.trainable = False

inputs = layers.Input(shape=IMG_SIZE+(3,))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.2)(x)
outputs = layers.Dense(10, activation='softmax')(x)
model = models.Model(inputs, outputs)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/efficientnet_v2/efficientnetv2-b0_notop.h
24274472/24274472 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

MODEL TRAINING

```
# MODEL TRAINING
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
model.compile(optimizer=tf.keras.optimizers.Adam(1e-4),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2)

history = model.fit(datatrain, validation_data=datavalid, epochs=15, callbacks=[early_stop, reduce_lr])
```

```
Epoch 1/15
76/76 ━━━━━━━━━━━━━━━━━━━━ 52s 180ms/step - accuracy: 0.2598 - loss: 2.1305 - val_accuracy: 0.7967 - val_loss: 1.1384 - lear
Epoch 2/15
76/76 ━━━━━━━━━━━━━━━━━━━━ 10s 137ms/step - accuracy: 0.7977 - loss: 0.9845 - val_accuracy: 0.9033 - val_loss: 0.5048 - lear
Epoch 3/15
76/76 ━━━━━━━━━━━━━━━━━━━━ 20s 123ms/step - accuracy: 0.8681 - loss: 0.4941 - val_accuracy: 0.9333 - val_loss: 0.3257 - lear
Epoch 4/15
76/76 ━━━━━━━━━━━━━━━━━━━━ 11s 140ms/step - accuracy: 0.9202 - loss: 0.3001 - val_accuracy: 0.9300 - val_loss: 0.2432 - lear
Epoch 5/15
76/76 ━━━━━━━━━━━━━━━━━━━━ 20s 130ms/step - accuracy: 0.9412 - loss: 0.2134 - val_accuracy: 0.9500 - val_loss: 0.1960 - lear
Epoch 6/15
76/76 ━━━━━━━━━━━━━━━━━━━━ 10s 125ms/step - accuracy: 0.9487 - loss: 0.1866 - val_accuracy: 0.9433 - val_loss: 0.1804 - lear
Epoch 7/15
76/76 ━━━━━━━━━━━━━━━━━━━━ 10s 123ms/step - accuracy: 0.9540 - loss: 0.1712 - val_accuracy: 0.9533 - val_loss: 0.1624 - lear
Epoch 8/15
76/76 ━━━━━━━━━━━━━━━━━━━━ 11s 142ms/step - accuracy: 0.9671 - loss: 0.1273 - val_accuracy: 0.9500 - val_loss: 0.1550 - lear
Epoch 9/15
76/76 ━━━━━━━━━━━━━━━━━━━━ 19s 123ms/step - accuracy: 0.9703 - loss: 0.1106 - val_accuracy: 0.9500 - val_loss: 0.1546 - lear
Epoch 10/15
76/76 ━━━━━━━━━━━━━━━━━━━━ 9s 121ms/step - accuracy: 0.9827 - loss: 0.0785 - val_accuracy: 0.9567 - val_loss: 0.1511 - learr
Epoch 11/15
76/76 ━━━━━━━━━━━━━━━━━━━━ 11s 134ms/step - accuracy: 0.9785 - loss: 0.0719 - val_accuracy: 0.9600 - val_loss: 0.1248 - lear
Epoch 12/15
76/76 ━━━━━━━━━━━━━━━━━━━━ 10s 133ms/step - accuracy: 0.9772 - loss: 0.0771 - val_accuracy: 0.9567 - val_loss: 0.1306 - lear
Epoch 13/15
76/76 ━━━━━━━━━━━━━━━━━━━━ 10s 131ms/step - accuracy: 0.9863 - loss: 0.0611 - val_accuracy: 0.9633 - val_loss: 0.1198 - lear
Epoch 14/15
76/76 ━━━━━━━━━━━━━━━━━━━━ 10s 123ms/step - accuracy: 0.9917 - loss: 0.0386 - val_accuracy: 0.9600 - val_loss: 0.1251 - lear
Epoch 15/15
76/76 ━━━━━━━━━━━━━━━━━━━━ 11s 142ms/step - accuracy: 0.9923 - loss: 0.0323 - val_accuracy: 0.9700 - val_loss: 0.1191 - lear
```

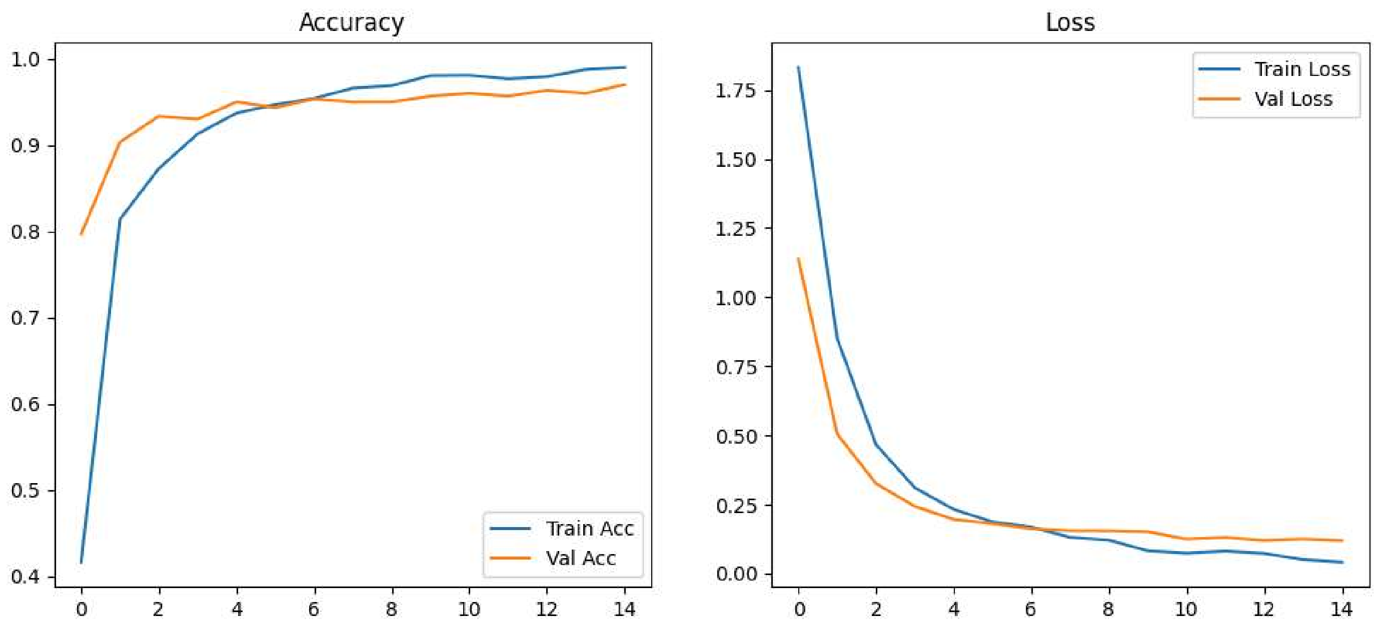
MODEL TUNING AND OPTIMIZATION

```
# MODEL TUNING AND OPTIMIZATION (optional fine-tune)
for layer in base_model.layers[100:]:
    layer.trainable = True
model.compile(optimizer=tf.keras.optimizers.Adam(1e-5),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(datatrain, validation_data=datavalid, epochs=5, callbacks=[early_stop, reduce_lr])
```

```
Epoch 1/5
76/76 ━━━━━━━━━━━━━━━━━━━━ 47s 181ms/step - accuracy: 0.9923 - loss: 0.0303 - val_accuracy: 0.9633 - val_loss: 0.1201 - lear
Epoch 2/5
76/76 ━━━━━━━━━━━━━━━━━━━━ 10s 134ms/step - accuracy: 0.9875 - loss: 0.0374 - val_accuracy: 0.9633 - val_loss: 0.1144 - lear
Epoch 3/5
76/76 ━━━━━━━━━━━━━━━━━━━━ 11s 138ms/step - accuracy: 0.9941 - loss: 0.0353 - val_accuracy: 0.9633 - val_loss: 0.1188 - lear
Epoch 4/5
76/76 ━━━━━━━━━━━━━━━━━━━━ 19s 121ms/step - accuracy: 0.9895 - loss: 0.0435 - val_accuracy: 0.9633 - val_loss: 0.1256 - lear
Epoch 5/5
76/76 ━━━━━━━━━━━━━━━━━━━━ 11s 135ms/step - accuracy: 0.9950 - loss: 0.0286 - val_accuracy: 0.9700 - val_loss: 0.1140 - lear
<keras.src.callbacks.history.History at 0x7d08ecd52850>
```

MODEL PERFORMANCE VISUALIZATION

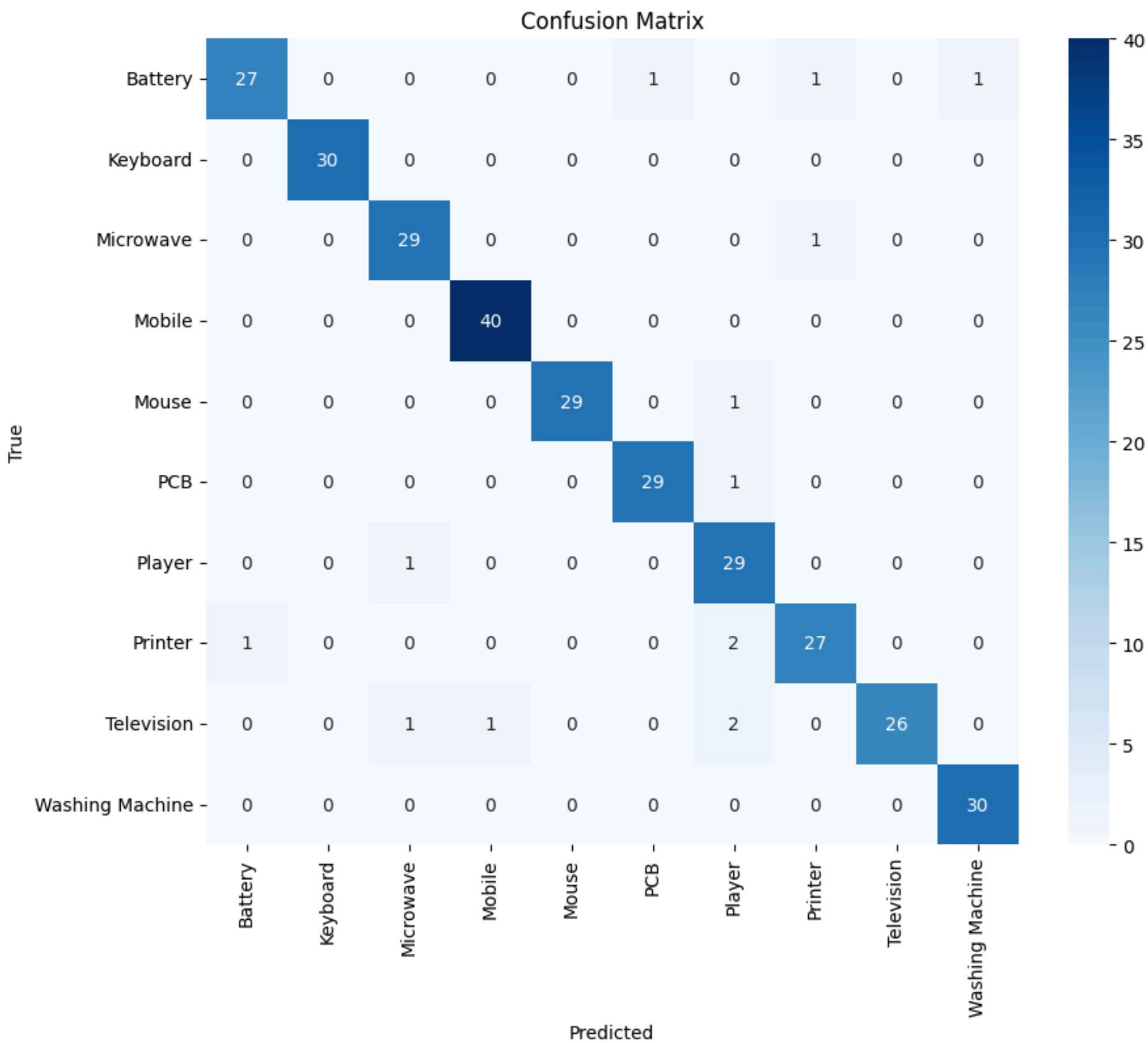
```
# MODEL PERFORMANCE VISUALIZATION
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title('Accuracy')
plt.legend()
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Loss')
plt.legend()
plt.show()
```



MODEL EVALUATION AND CONFUSION MATRIX

```
#  MODEL EVALUATION
test_loss, test_acc = model.evaluate(datatest)
print(f"Test Accuracy: {test_acc:.4f}, Test Loss: {test_loss:.4f}")

y_true = np.concatenate([y.numpy() for _, y in datatest], axis=0)
y_pred = np.argmax(model.predict(datatest), axis=1)

cm = confusion_matrix(y_true, y_pred)
print(classification_report(y_true, y_pred, target_names=class_names))

plt.figure(figsize=(10,8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 128ms/step - accuracy: 0.9555 - loss: 0.1602
✅ Test Accuracy: 0.9548, Test Loss: 0.1317
10/10 ━━━━━━━━━━━━━━━━━━━━ 7s 393ms/step
                 precision    recall  f1-score   support

        Battery       0.96      0.90      0.93        30
       Keyboard       1.00      1.00      1.00        30
      Microwave       0.94      0.97      0.95        30
         Mobile       0.98      1.00      0.99        40
          Mouse       1.00      0.97      0.98        30
            PCB       0.97      0.97      0.97        30
         Player       0.83      0.97      0.89        30
        Printer       0.93      0.90      0.92        30
     Television       1.00      0.87      0.93        30
Washing Machine       0.97      1.00      0.98        30

       accuracy                           0.95       310
      macro avg       0.96      0.95      0.95       310
   weighted avg       0.96      0.95      0.95       310
```



True \ Predicted	Battery	Keyboard	Microwave	Mobile	Mouse	PCB	Player	Printer	Television	Washing Machine
Battery	27	0	0	0	0	1	0	1	0	1
Keyboard	0	30	0	0	0	0	0	0	0	0
Microwave	0	0	29	0	0	0	0	1	0	0
Mobile	0	0	0	40	0	0	0	0	0	0
Mouse	0	0	0	0	29	0	1	0	0	0
PCB	0	0	0	0	0	29	1	0	0	0
Player	0	0	1	0	0	0	29	0	0	0
Printer	1	0	0	0	0	0	2	27	0	0
Television	0	0	1	1	0	0	2	0	26	0
Washing Machine	0	0	0	0	0	0	0	0	0	30

FINAL TESTING AND SAVE THE MODEL

```
# FINAL TESTING AND SAVE THE MODEL
model.save('efficientnetv2b0_ewaste_final.keras')
print("Keras model saved")
```

```
Keras model saved
```

Save TFLite

```
# Save TFLite
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()
with open('efficientnetv2b0_ewaste_final.tflite', 'wb') as f:
    f.write(tflite_model)
print("TFLite model saved")
```

```
  137479625108048: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625107280: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625104976: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625109008: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625109200: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625107472: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625109584: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625105552: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625110736: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625111504: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625110352: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625110160: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625110928: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625112848: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625113232: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625109392: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625112656: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625572432: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625573008: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625574544: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625573776: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625573584: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625575504: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625575696: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625573968: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625576080: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625573200: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625577232: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625578000: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625576848: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625576656: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625577424: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625579344: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625580304: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625579536: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625578192: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625581264: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625580880: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625582224: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625581456: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625579152: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625583184: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625583376: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625581648: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625583760: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625579728: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625584912: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625585680: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625584528: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625584336: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625585104: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625587024: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625587984: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625587216: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625585872: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479625586832: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137479624263248: TensorSpec(shape=(), dtype=tf.resource, name=None)
TFLite model saved
```

Predictions on sample test images

```
# Show predictions on sample test images
for images, labels in datatest.take(1):
    preds = model.predict(images)
    pred_classes = tf.argmax(preds, axis=1)
    for i in range(8):
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(f"True: {class_names[labels[i]]}, Pred: {class_names[pred_classes[i]]}")
        plt.axis("off")
        plt.show()
```

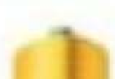
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step

True: Battery, Pred: Battery

True: Battery, Pred: Printer

True: Battery, Pred: Battery

True: Battery, Pred: Battery



True: Battery, Pred: Battery

True: Battery, Pred: Battery

True: Battery, Pred: PCB

True: Battery, Pred: Battery

Using CNN Model

```
# Normal CNN Model
normal_model = tf.keras.Sequential([
    tf.keras.layers.Rescaling(1./255, input_shape=IMG_SIZE+(3,)),  # Simple rescaling
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(128, 3, activation='relu'),
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```