



✓ Garbage Classification with EfficientNetV2B2

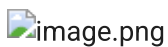
Project Description

In this project, we aim to develop a sophisticated **garbage classification system** leveraging the **EfficientNetV2B2** architecture. Our primary dataset serves as a foundation for building models that can eventually automate waste segregation, a critical step in optimizing recycling and waste management, ultimately aiding in environmental conservation.

Goal: To develop an accurate and efficient garbage classification model using EfficientNetV2B2 and transfer learning for automated waste sorting.

Challenges and Scope

Key Challenge: A notable challenge encountered is the inherent **class imbalance** within the dataset.



Transfer Learning is a machine learning technique where a pre-trained model developed for a speci

Benefits

- **Reduces training time** — you don't start from scratch.
 - **Leverages learned features** from large datasets (like ImageNet).
 - **Improves performance**, especially with limited data.
-

How Does It Work?

1. Load a pretrained model (e.g., ResNet, EfficientNet).
2. **Freeze** the pretrained layers (optional).
3. Add new layers for your custom task.
4. Train on your new dataset (can also fine-tune).

EfficientNetV2B2: Transfer Learning Backbone

EfficientNetV2B2 is a mid-sized model from the EfficientNetV2 family developed by **Google**, balancing performance and efficiency.

⚙️ Key Features:

- **Fused MBConv blocks** — enhance both training stability and speed.

- **Progressive learning** – enables better generalization with less computation.
- **Improved architecture** – achieves higher accuracy with optimized FLOPs.

Why Use EfficientNetV2B2?

Feature	Description
Balanced Performance	Great trade-off between speed and accuracy
Scalable	Suitable for moderately complex datasets
Pretrained on ImageNet	Solid backbone for transfer learning tasks
Efficient	Faster convergence with fewer resources needed

✓ Core Libraries

- `tensorflow`: For deep learning model building and training.
- `numpy`: For numerical operations and array manipulation.
- `matplotlib.pyplot`: For plotting training curves and results.

```
import numpy as np # Importing NumPy for numerical operations and array manipulations
import matplotlib.pyplot as plt # Importing Matplotlib for plotting graphs and visualizations
import seaborn as sns # Importing Seaborn for statistical data visualization, built on top of Matplotlib
import tensorflow as tf # Importing TensorFlow for building and training machine learning models
from tensorflow import keras # Importing Keras, a high-level API for TensorFlow, to simplify model building
from tensorflow.keras import Layer # Importing Layer class for creating custom layers in Keras
from tensorflow.keras.models import Sequential # Importing Sequential model for building neural networks
from tensorflow.keras.layers import Rescaling, GlobalAveragePooling2D
from tensorflow.keras import layers, optimizers, callbacks # Importing various modules for layers, optimizers, and callbacks
from sklearn.utils.class_weight import compute_class_weight # Importing function to compute class weights
from tensorflow.keras.applications import EfficientNetV2B2 # Importing EfficientNetV2S model for transfer learning
from sklearn.metrics import confusion_matrix, classification_report # Importing functions to evaluate model performance
import gradio as gr # Importing Gradio for creating interactive web interfaces for machine learning
```

✓ 1. Explore and Understand the Data

- Load image dataset using tools like `image_dataset_from_directory`.
- Visualize sample images from each class.
- Check the number of images per class to ensure balance.
- Understand image dimensions, color channels, and class labels.

✓ Load image dataset using tools like `image_dataset_from_directory`.

Split data into training, validation, and testing sets.

```
tf.keras.utils.image_dataset_from_directory(...)
```

Used to load images from a directory where each subfolder represents a class.

path

Root directory path containing one subdirectory per class.

shuffle=True

Randomly shuffles the image data. Useful during training to prevent the model from learning the order of the data.

image_size=(128, 128)

Resizes all loaded images to this target size (width, height).

This must match the input size expected by the model.

batch_size=32

Number of images per batch during training.

This affects memory usage and the frequency of model updates.

validation_split=False

If set to a float (e.g., 0.2), splits a portion of the data for validation.

If False, no split is applied.

```
dataset_dir= r"C:\Users\Edunet Foundation\Downloads\project\garbage\TrashType_Image_Dataset"
image_size = (124, 124)
batch_size = 32
seed = 42
```

```
train_ds = tf.keras.utils.image_dataset_from_directory(
    dataset_dir,
    validation_split=0.2,
    subset="training",
    seed=seed,
    shuffle = True,
    image_size=image_size,
    batch_size=batch_size
)
```

🔄 Found 2527 files belonging to 6 classes.
Using 2022 files for training.

```
val_ds = tf.keras.utils.image_dataset_from_directory(
    dataset_dir,
    validation_split=0.2,
    subset="validation",
    seed=seed,
    shuffle = True,
    image_size=image_size,
    batch_size=batch_size
)
val_class= val_ds.class_names
```

🔄 Found 2527 files belonging to 6 classes.
Using 505 files for validation.

```
# Get the total number of batches in the validation dataset
val_batches = tf.data.experimental.cardinality(val_ds)
```

```
# Split the validation dataset into two equal parts:
# First half becomes the test dataset
test_ds = val_ds.take(val_batches // 2)
```

```
# Second half remains as the validation dataset
val_dat = val_ds.skip(val_batches // 2)

# Optimize test dataset by caching and prefetching to improve performance
test_ds_eval = test_ds.cache().prefetch(tf.data.AUTOTUNE)
```

```
print(train_ds.class_names)
print(val_class)
print(len(train_ds.class_names))
```

```
⇒ ['cardboard', 'glass', 'metal', 'paper', 'plastic', 'trash']
   ['cardboard', 'glass', 'metal', 'paper', 'plastic', 'trash']
   6
```

✓ Visualize sample images from each class.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(12):
        ax = plt.subplot(4, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(train_ds.class_names[labels[i]])
        plt.axis("off")
```



glass



plastic



glass



metal



plastic



metal



cardboard



plastic



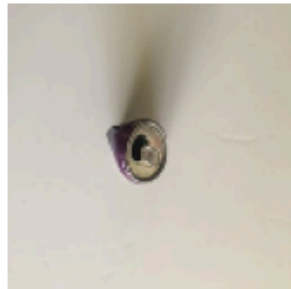
plastic



paper



metal



paper



- ✓ Check the number of images per class to ensure balance
- Understand image properties like Image dimensions, Class labels

```
def count_distribution(dataset, class_names):  
    total = 0  
    counts = {name: 0 for name in class_names}  
  
    for _, labels in dataset:  
        for label in labels.numpy():  
            class_name = class_names[label]
```

```

        counts[class_name] += 1
    total += 1

    for k in counts:
        counts[k] = round((counts[k] / total) * 100, 2) # Convert to percentage
    return counts

# Function to plot class distribution
def simple_bar_plot(dist, title):
    plt.bar(dist.keys(), dist.values(), color='cornflowerblue')
    plt.title(title)
    plt.ylabel('Percentage (%)')
    plt.xticks(rotation=45)
    plt.ylim(0, 100)
    plt.tight_layout()
    plt.show()

class_names = train_ds.class_names

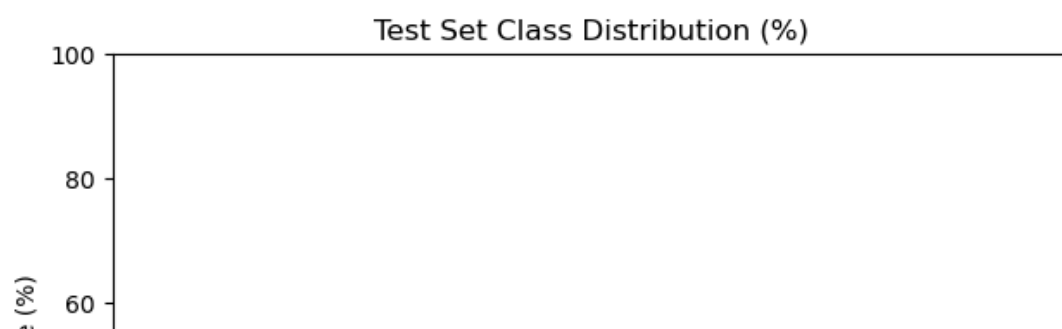
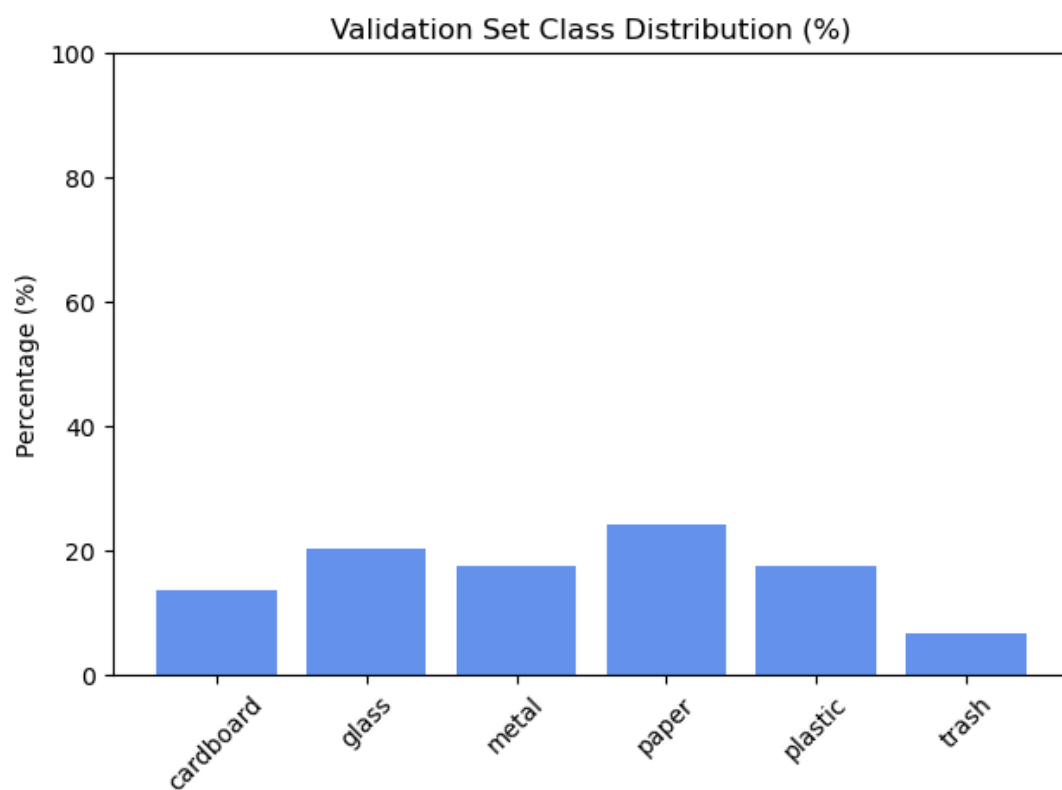
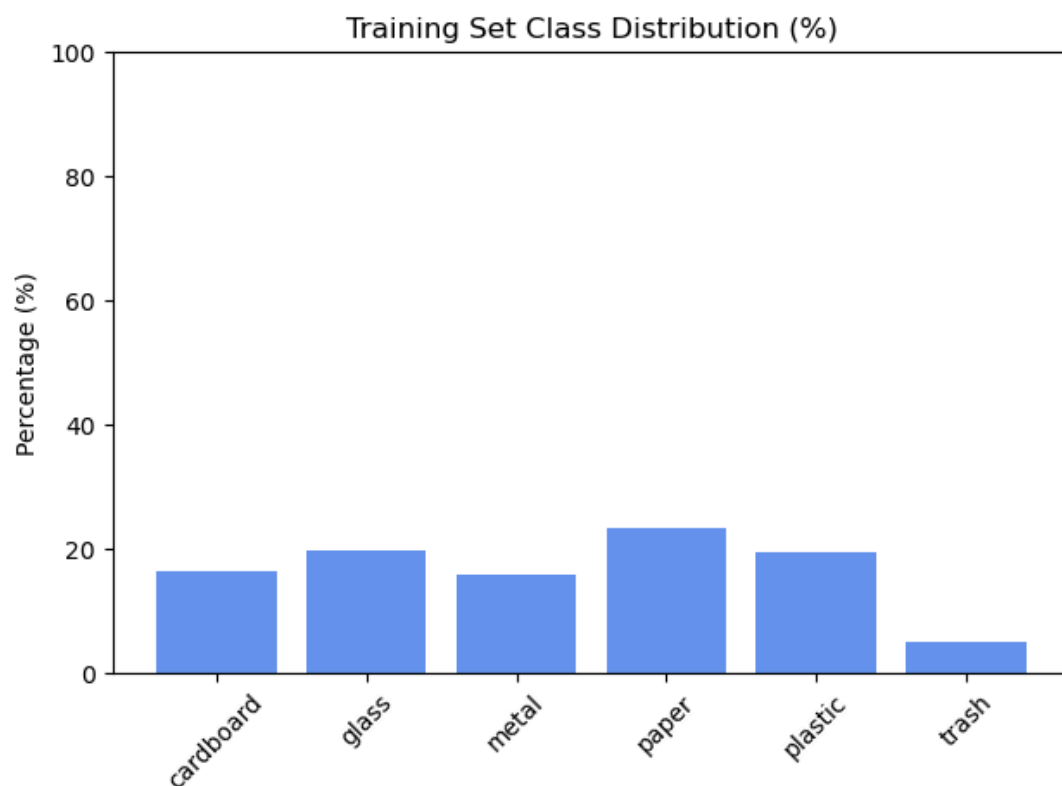
# Get class distributions
train_dist = count_distribution(train_ds, class_names)
val_dist = count_distribution(val_ds, class_names)
test_dist = count_distribution(test_ds, class_names)
overall_dist = {}
for k in class_names:
    overall_dist[k] = round((train_dist[k] + val_dist[k]) / 2, 2)

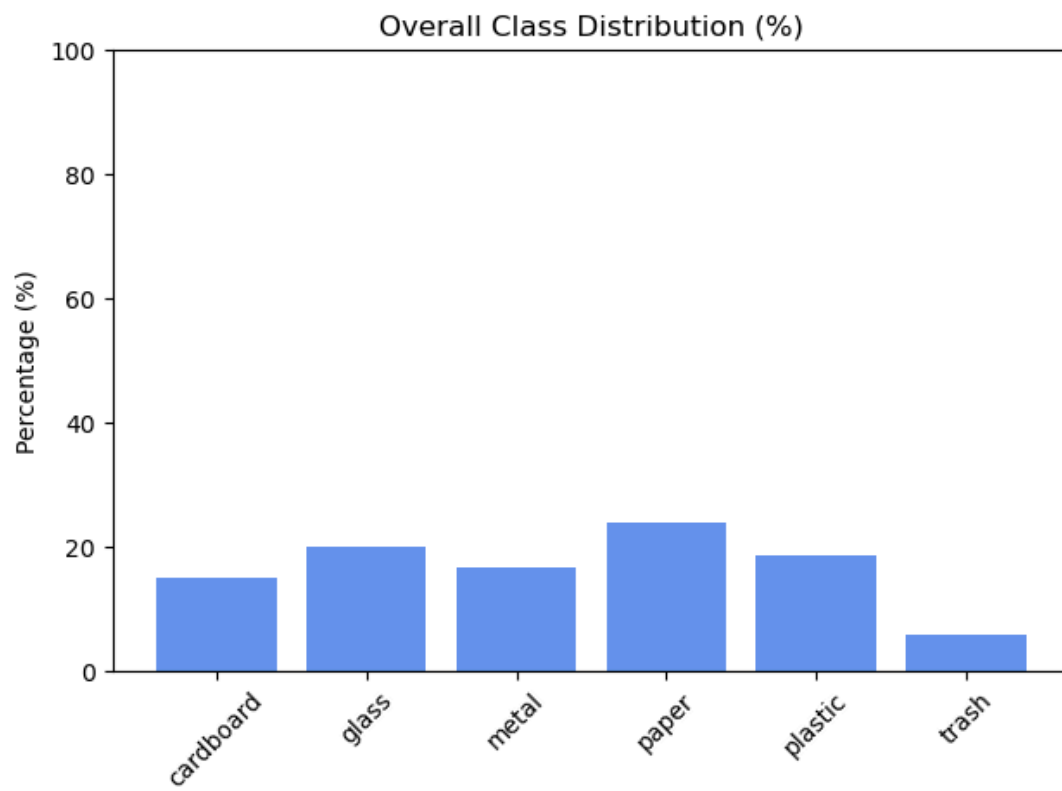
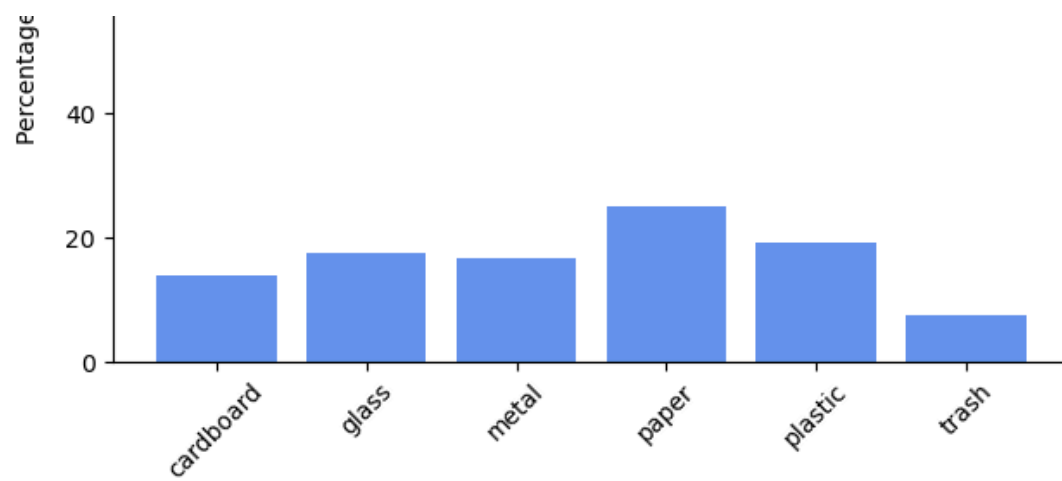
print(train_dist)
print(val_dist)
print(test_dist)
print(overall_dist)

🔗 {'cardboard': 16.52, 'glass': 19.73, 'metal': 15.92, 'paper': 23.29, 'plastic': 19.44, 'trash': 16.14}
🔗 {'cardboard': 13.66, 'glass': 20.2, 'metal': 17.43, 'paper': 24.36, 'plastic': 17.62, 'trash': 15.99}
🔗 {'cardboard': 14.06, 'glass': 17.58, 'metal': 16.8, 'paper': 25.0, 'plastic': 19.14, 'trash': 16.14}
🔗 {'cardboard': 15.09, 'glass': 19.96, 'metal': 16.68, 'paper': 23.82, 'plastic': 18.53, 'trash': 16.14}

# Show visualizations
simple_bar_plot(train_dist, "Training Set Class Distribution (%)")
simple_bar_plot(val_dist, "Validation Set Class Distribution (%)")
simple_bar_plot(test_dist, "Test Set Class Distribution (%)")
simple_bar_plot(overall_dist, "Overall Class Distribution (%)")

```





✓ Inference on Class Imbalance

The "**Garbage Image Dataset**" reveals a noticeable **imbalance** in the distribution of its image categories:

Category	Image Count	Updated Distribution
Cardboard	403	15.09
Glass	501	19.96
Metal	410	16.68
Paper	594	23.82
Plastic	482	18.53
Trash	137	5.91

Analogy:

Imagine teaching a child to identify animals by showing them **95 pictures of cats** and just **5 pictures of dogs**.

They'd probably think **most pets are cats**, right?

Similarly, our model sees a lot of "**paper**" and very little "**trash**", which **biases** its understanding.

Key Problems Caused by Class Imbalance:

1 Bias

- The model may **overpredict common classes** like "paper" and **underpredict rare ones** like "trash".

2 Generalization Issues

- If the real-world distribution is more balanced, the model may **fail to generalize** and **misclassify rare classes**.

3 Accuracy Deception

- The model might appear to have **high overall accuracy** just by **predicting the majority class**, while **failing** on underrepresented ones.

Solution Approaches :

- Use **class weights** to handle imbalanced data in training,
- Apply **data augmentation** to increase training data diversity

Conclusion: Always check class distribution. A seemingly "accurate" model might just be **biased** toward the dominant class.

✓ Addressing Imbalance Using Class Weights:

To tackle our imbalanced image dataset, we'll utilize **class weights**. These weights assign more importance to underrepresented classes during training. The weights are computed inversely proportional to class frequencies using utilities like `compute_class_weight` from **scikit-learn**, based on the distribution of images in each class. The formula is:

$$\text{weight}(\text{class}) = \frac{\text{total samples}}{\text{number of classes} \times \text{samples in that class}}$$


These computed weights are then passed to the model.

```
# Count class occurrences and prepare label list
class_counts = {i: 0 for i in range(len(class_names))}
all_labels = []

for images, labels in train_ds:
    for label in labels.numpy():
        class_counts[label] += 1
        all_labels.append(label)

# Compute class weights (index aligned)
class_weights_array = compute_class_weight(
    class_weight='balanced',
    classes=np.arange(len(class_names)),
    y=all_labels
)

# Create dictionary mapping class index to weight
class_weights = {i: w for i, w in enumerate(class_weights_array)}
```

```
#  Optional: print results
print("Class Counts:", class_counts)
print("Class Weights:", class_weights)
```

```
➡ Class Counts: {0: 334, 1: 399, 2: 322, 3: 471, 4: 393, 5: 103}
   Class Weights: {0: 1.0089820359281436, 1: 0.8446115288220551, 2: 1.046583850931677, 3: 0.71549...
```

✓ 2. Data Preprocessing / Preparation

- Resize and rescale images.
- Apply data augmentation (e.g., RandomFlip, RandomRotation, RandomZoom) to improve generalization.
- Normalize images (using preprocess_input if using pre-trained models like EfficientNet).

```
# Define data augmentation pipeline
data_augmentation = Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
    layers.RandomContrast(0.1),
])
```

✓ 3. Model Selection

- Choose a base model: Custom CNN or Transfer Learning (e.g., EfficientNetV2B2).

- Decide whether to use pre-trained weights (e.g., ImageNet).
- Define whether layers should be trainable or frozen during initial training.

```
# Load the pretrained MobileNetV3Small model (without the top classification layer)
base_model = EfficientNetV2B2(include_top=False, input_shape=(124, 124, 3), include_preprocessing=T

# Freeze early layers (to retain general pretrained features)
base_model.trainable = True
for layer in base_model.layers[:100]: # You can adjust this number
    layer.trainable = False
```

✓ 4. Model Training

- Build the model architecture using `Sequential` or Functional API.
- Compile the model with loss function (`sparse_categorical_crossentropy`), optimizer (e.g., `Adam`), and evaluation metrics (`accuracy`).

5. Model Tuning and Optimization

- Tune hyperparameters: learning rate, batch size, number of layers, dropout rate.
- Use callbacks: `EarlyStopping`,
- Optionally perform fine-tuning on pre-trained models by unfreezing some layers.

✓ Model Architecture and Layer Utilities

- **Sequential**: A simple way to build models by stacking layers one after the other in a linear fashion.
- **RandomFlip**: A data augmentation layer that flips input images horizontally or vertically at random, helping the model generalize better.
- **RandomRotation**: Randomly rotates images by a specified angle range during training to make the model invariant to orientation.
- **RandomZoom**: Applies random zoom-in or zoom-out to training images, helping the model recognize objects at various scales.
- **Dropout**: A regularization method that randomly "drops" (sets to zero) a fraction of input units during training to prevent overfitting.
- **GlobalAveragePooling2D**: Reduces each feature map to a single number by taking the average, reducing model parameters and helping prevent overfitting.
- **Dense**: A fully connected neural network layer used to learn complex features and typically found at the end of the model for classification.
- **Input**: Specifies the input shape and data type for the model; acts as the starting point of the model architecture.

- **EfficientNetV2B2**: A pre-trained convolutional neural network from the EfficientNetV2 family, known for being lightweight and high-performing, commonly used for transfer learning.

```
# Build the final model
model = Sequential([
    layers.Input(shape=(124, 124, 3)),
    data_augmentation,
    base_model,
    GlobalAveragePooling2D(),
    layers.Dropout(0.3),
    layers.Dense(6, activation='softmax') # Change to your number of classes
])
```

```
# ⚙️ Compile the model
model.compile(
    optimizer=optimizers.Adam(learning_rate=1e-4),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

▼ Callbacks

- **EarlyStopping**: To stop training when validation performance stops improving

```
# Define an EarlyStopping callback to stop training when validation loss stops improving
early = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',           # Metric to monitor (validation loss here)
    patience=3,                  # Number of epochs to wait after last improvement before stopping
    restore_best_weights=True     # After stopping, restore the model weights from the epoch with
)
```

▼ Train the model using `.fit()` with appropriate epochs, batch_size, and callbacks like EarlyStopping.

```
# Set the number of epochs to train the model
epochs = 15 # Number of times the model will go through the entire dataset

# Train the model using the fit function
history = model.fit(
    train_ds,                    # Training dataset used to adjust model weights
    validation_data=val_ds,      # Validation dataset to monitor performance on unseen data
    epochs=epochs,               # Number of training cycles, referencing the variable set earlier
    class_weight=class_weights, # Handles class imbalances by assigning appropriate weights
    batch_size=32,               # Number of samples processed in each training step
    callbacks=[early]            # Implements early stopping to prevent unnecessary training
)
```

```
↔ Epoch 1/15
64/64 ————— 317s 2s/step - accuracy: 0.2712 - loss: 1.7355 - val_accuracy: 0.2712
Epoch 2/15
```

```

64/64 ————— 140s 2s/step - accuracy: 0.6397 - loss: 1.1280 - val_accuracy: 0.6397
Epoch 3/15
64/64 ————— 140s 2s/step - accuracy: 0.7807 - loss: 0.7128 - val_accuracy: 0.7807
Epoch 4/15
64/64 ————— 139s 2s/step - accuracy: 0.8405 - loss: 0.5068 - val_accuracy: 0.8405
Epoch 5/15
64/64 ————— 141s 2s/step - accuracy: 0.8794 - loss: 0.3979 - val_accuracy: 0.8794
Epoch 6/15
64/64 ————— 142s 2s/step - accuracy: 0.9026 - loss: 0.2915 - val_accuracy: 0.9026
Epoch 7/15
64/64 ————— 201s 2s/step - accuracy: 0.9277 - loss: 0.2199 - val_accuracy: 0.9277
Epoch 8/15
64/64 ————— 141s 2s/step - accuracy: 0.9349 - loss: 0.1977 - val_accuracy: 0.9349
Epoch 9/15
64/64 ————— 141s 2s/step - accuracy: 0.9516 - loss: 0.1733 - val_accuracy: 0.9516
Epoch 10/15
64/64 ————— 148s 2s/step - accuracy: 0.9542 - loss: 0.1357 - val_accuracy: 0.9542
Epoch 11/15
64/64 ————— 195s 2s/step - accuracy: 0.9591 - loss: 0.1234 - val_accuracy: 0.9591
Epoch 12/15
64/64 ————— 141s 2s/step - accuracy: 0.9787 - loss: 0.0861 - val_accuracy: 0.9787
Epoch 13/15
64/64 ————— 141s 2s/step - accuracy: 0.9735 - loss: 0.1004 - val_accuracy: 0.9735
Epoch 14/15
64/64 ————— 143s 2s/step - accuracy: 0.9642 - loss: 0.1091 - val_accuracy: 0.9642
Epoch 15/15
64/64 ————— 141s 2s/step - accuracy: 0.9698 - loss: 0.0963 - val_accuracy: 0.9698

```

📝 Summary (optional but useful)
model.summary()

➡ Model: "sequential_1"

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 124, 124, 3)	0
efficientnetv2-b2 (Functional)	(None, 4, 4, 1408)	8,769,374
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1408)	0
dropout (Dropout)	(None, 1408)	0
dense (Dense)	(None, 6)	8,454

```

Total params: 24,727,114 (94.33 MB)
Trainable params: 7,974,642 (30.42 MB)
Non-trainable params: 803,186 (3.06 MB)
Optimizer params: 15,949,286 (60.84 MB)

```

base_model.summary() # Print the architecture summary of the base model

Model: "efficientnetv2-b2"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 124, 124, 3)	0	-
rescaling (Rescaling)	(None, 124, 124, 3)	0	input_layer[0][0]
normalization (Normalization)	(None, 124, 124, 3)	0	rescaling[0][0]
stem_conv (Conv2D)	(None, 62, 62, 32)	864	normalization[0]...
stem_bn (BatchNormalizatio...	(None, 62, 62, 32)	128	stem_conv[0][0]
stem_activation (Activation)	(None, 62, 62, 32)	0	stem_bn[0][0]
block1a_project_co... (Conv2D)	(None, 62, 62, 16)	4,608	stem_activation[...
block1a_project_bn (BatchNormalizatio...	(None, 62, 62, 16)	64	block1a_project_...
block1a_project_ac... (Activation)	(None, 62, 62, 16)	0	block1a_project_...
block1b_project_co... (Conv2D)	(None, 62, 62, 16)	2,304	block1a_project_...
block1b_project_bn (BatchNormalizatio...	(None, 62, 62, 16)	64	block1b_project_...
block1b_project_ac... (Activation)	(None, 62, 62, 16)	0	block1b_project_...
block1b_drop (Dropout)	(None, 62, 62, 16)	0	block1b_project_...
block1b_add (Add)	(None, 62, 62, 16)	0	block1b_drop[0][...] block1a_project_...
block2a_expand_conv (Conv2D)	(None, 31, 31, 64)	9,216	block1b_add[0][0]
block2a_expand_bn (BatchNormalizatio...	(None, 31, 31, 64)	256	block2a_expand_c...
block2a_expand_act... (Activation)	(None, 31, 31, 64)	0	block2a_expand_b...
block2a_project_co... (Conv2D)	(None, 31, 31, 32)	2,048	block2a_expand_a...
block2a_project_bn (BatchNormalizatio...	(None, 31, 31, 32)	128	block2a_project_...

block2b_expand_conv (Conv2D)	(None, 31, 31, 128)	36,864	block2a_project_...
block2b_expand_bn (BatchNormalizatio...	(None, 31, 31, 128)	512	block2b_expand_c...
block2b_expand_act... (Activation)	(None, 31, 31, 128)	0	block2b_expand_b...
block2b_project_co... (Conv2D)	(None, 31, 31, 32)	4,096	block2b_expand_a...
block2b_project_bn (BatchNormalizatio...	(None, 31, 31, 32)	128	block2b_project_...
block2b_drop (Dropout)	(None, 31, 31, 32)	0	block2b_project_...
block2b_add (Add)	(None, 31, 31, 32)	0	block2b_drop[0][... block2a_project_...
block2c_expand_conv (Conv2D)	(None, 31, 31, 128)	36,864	block2b_add[0][0]
block2c_expand_bn (BatchNormalizatio...	(None, 31, 31, 128)	512	block2c_expand_c...
block2c_expand_act... (Activation)	(None, 31, 31, 128)	0	block2c_expand_b...
block2c_project_co... (Conv2D)	(None, 31, 31, 32)	4,096	block2c_expand_a...
block2c_project_bn (BatchNormalizatio...	(None, 31, 31, 32)	128	block2c_project_...
block2c_drop (Dropout)	(None, 31, 31, 32)	0	block2c_project_...
block2c_add (Add)	(None, 31, 31, 32)	0	block2c_drop[0][... block2b_add[0][0]
block3a_expand_conv (Conv2D)	(None, 16, 16, 128)	36,864	block2c_add[0][0]
block3a_expand_bn (BatchNormalizatio...	(None, 16, 16, 128)	512	block3a_expand_c...
block3a_expand_act... (Activation)	(None, 16, 16, 128)	0	block3a_expand_b...
block3a_project_co... (Conv2D)	(None, 16, 16, 56)	7,168	block3a_expand_a...
block3a_project_bn (BatchNormalizatio...	(None, 16, 16, 56)	224	block3a_project_...
block3b_expand_conv (Conv2D)	(None, 16, 16, 224)	112,896	block3a_project_...

(Conv2D)	(None, 16, 16, 224)	896	block3b_expand_bn
block3b_expand_bn (BatchNormalization)	(None, 16, 16, 224)	0	block3b_expand_act...
block3b_expand_act... (Activation)	(None, 16, 16, 224)	12,544	block3b_project_co...
block3b_project_co... (Conv2D)	(None, 16, 16, 56)	224	block3b_project_bn
block3b_project_bn (BatchNormalization)	(None, 16, 16, 56)	0	block3b_drop (Dropout)
block3b_drop (Dropout)	(None, 16, 16, 56)	0	block3b_add (Add)
block3b_add (Add)	(None, 16, 16, 56)	112,896	block3c_expand_conv (Conv2D)
block3c_expand_conv (Conv2D)	(None, 16, 16, 224)	896	block3c_expand_bn (BatchNormalization)
block3c_expand_bn (BatchNormalization)	(None, 16, 16, 224)	0	block3c_expand_act...
block3c_expand_act... (Activation)	(None, 16, 16, 224)	12,544	block3c_project_co...
block3c_project_co... (Conv2D)	(None, 16, 16, 56)	224	block3c_project_bn
block3c_project_bn (BatchNormalization)	(None, 16, 16, 56)	0	block3c_drop (Dropout)
block3c_drop (Dropout)	(None, 16, 16, 56)	0	block3c_add (Add)
block3c_add (Add)	(None, 16, 16, 56)	12,544	block4a_expand_conv (Conv2D)
block4a_expand_conv (Conv2D)	(None, 16, 16, 224)	896	block4a_expand_bn (BatchNormalization)
block4a_expand_bn (BatchNormalization)	(None, 16, 16, 224)	0	block4a_expand_act...
block4a_expand_act... (Activation)	(None, 16, 16, 224)	2,016	block4a_dwconv2 (DepthwiseConv2D)
block4a_dwconv2 (DepthwiseConv2D)	(None, 8, 8, 224)	896	block4a_bn (BatchNormalization)
block4a_bn (BatchNormalization)	(None, 8, 8, 224)	0	block4a_activation (Activation)
block4a_activation (Activation)	(None, 8, 8, 224)	0	block4a_se.squeeze
block4a_se.squeeze	(None, 224)		

(GlobalAveragePool...				
block4a_se_reshape (Reshape)	(None, 1, 1, 224)	0	block4a_se_squee...	
block4a_se_reduce (Conv2D)	(None, 1, 1, 14)	3,150	block4a_se_resha...	
block4a_se_expand (Conv2D)	(None, 1, 1, 224)	3,360	block4a_se_reduc...	
block4a_se_excite (Multiply)	(None, 8, 8, 224)	0	block4a_activati...	block4a_se_expan...
block4a_project_co...	(None, 8, 8, 104)	23,296	block4a_se_excit...	
block4a_project_bn (BatchNormalizatio...	(None, 8, 8, 104)	416	block4a_project_...	
block4b_expand_conv (Conv2D)	(None, 8, 8, 416)	43,264	block4a_project_...	
block4b_expand_bn (BatchNormalizatio...	(None, 8, 8, 416)	1,664	block4b_expand_c...	
block4b_expand_act...	(None, 8, 8, 416)	0	block4b_expand_b...	
block4b_dwconv2 (DepthwiseConv2D)	(None, 8, 8, 416)	3,744	block4b_expand_a...	
block4b_bn (BatchNormalizatio...	(None, 8, 8, 416)	1,664	block4b_dwconv2[...	
block4b_activation (Activation)	(None, 8, 8, 416)	0	block4b_bn[0][0]	
block4b_se_squeeze (GlobalAveragePool...	(None, 416)	0	block4b_activati...	
block4b_se_reshape (Reshape)	(None, 1, 1, 416)	0	block4b_se_squee...	
block4b_se_reduce (Conv2D)	(None, 1, 1, 26)	10,842	block4b_se_resha...	
block4b_se_expand (Conv2D)	(None, 1, 1, 416)	11,232	block4b_se_reduc...	
block4b_se_excite (Multiply)	(None, 8, 8, 416)	0	block4b_activati...	block4b_se_expan...
block4b_project_co...	(None, 8, 8, 104)	43,264	block4b_se_excit...	
block4b_project_bn (BatchNormalizatio...	(None, 8, 8, 104)	416	block4b_project_...	
block4b_drop	(None, 8, 8, 104)	0	block4b_project_...	

(Dropout)				
block4b_add (Add)	(None, 8, 8, 104)	0	block4b_drop[0][...]	block4a_project...
block4c_expand_conv (Conv2D)	(None, 8, 8, 416)	43,264	block4b_add[0][0]	
block4c_expand_bn (BatchNormalizatio...	(None, 8, 8, 416)	1,664	block4c_expand_c...	
block4c_expand_act...	(None, 8, 8, 416)	0	block4c_expand_b...	
block4c_dwconv2 (DepthwiseConv2D)	(None, 8, 8, 416)	3,744	block4c_expand_a...	
block4c_bn (BatchNormalizatio...	(None, 8, 8, 416)	1,664	block4c_dwconv2[...]	
block4c_activation (Activation)	(None, 8, 8, 416)	0	block4c_bn[0][0]	
block4c_se_squeeze (GlobalAveragePool...	(None, 416)	0	block4c_activati...	
block4c_se_reshape (Reshape)	(None, 1, 1, 416)	0	block4c_se_squee...	
block4c_se_reduce (Conv2D)	(None, 1, 1, 26)	10,842	block4c_se_resha...	
block4c_se_expand (Conv2D)	(None, 1, 1, 416)	11,232	block4c_se_reduc...	
block4c_se_excite (Multiply)	(None, 8, 8, 416)	0	block4c_activati...	block4c_se_expan...
block4c_project_co... (Conv2D)	(None, 8, 8, 104)	43,264	block4c_se_excit...	
block4c_project_bn (BatchNormalizatio...	(None, 8, 8, 104)	416	block4c_project_...	
block4c_drop (Dropout)	(None, 8, 8, 104)	0	block4c_project_...	
block4c_add (Add)	(None, 8, 8, 104)	0	block4c_drop[0][...]	block4b_add[0][0]
block4d_expand_conv (Conv2D)	(None, 8, 8, 416)	43,264	block4c_add[0][0]	
block4d_expand_bn (BatchNormalizatio...	(None, 8, 8, 416)	1,664	block4d_expand_c...	
block4d_expand_act...	(None, 8, 8, 416)	0	block4d_expand_b...	
block4d_dwconv2	(None, 8, 8, 416)	3,744	block4d_expand_a...	

(DepthwiseConv2D)				
block4d_bn (BatchNormalizatio...	(None, 8, 8, 416)	1,664	block4d_dwconv2[...	
block4d_activation (Activation)	(None, 8, 8, 416)	0	block4d_bn[0][0]	
block4d_se_squeeze (GlobalAveragePool...	(None, 416)	0	block4d_activati...	
block4d_se_reshape (Reshape)	(None, 1, 1, 416)	0	block4d_se_squee...	
block4d_se_reduce (Conv2D)	(None, 1, 1, 26)	10,842	block4d_se_resha...	
block4d_se_expand (Conv2D)	(None, 1, 1, 416)	11,232	block4d_se_reduc...	
block4d_se_excite (Multiply)	(None, 8, 8, 416)	0	block4d_activati... block4d_se_expan...	
block4d_project_co... (Conv2D)	(None, 8, 8, 104)	43,264	block4d_se_excit...	
block4d_project_bn (BatchNormalizatio...	(None, 8, 8, 104)	416	block4d_project_...	
block4d_drop (Dropout)	(None, 8, 8, 104)	0	block4d_project_...	
block4d_add (Add)	(None, 8, 8, 104)	0	block4d_drop[0][... block4c_add[0][0]	
block5a_expand_conv (Conv2D)	(None, 8, 8, 624)	64,896	block4d_add[0][0]	
block5a_expand_bn (BatchNormalizatio...	(None, 8, 8, 624)	2,496	block5a_expand_c...	
block5a_expand_act... (Activation)	(None, 8, 8, 624)	0	block5a_expand_b...	
block5a_dwconv2 (DepthwiseConv2D)	(None, 8, 8, 624)	5,616	block5a_expand_a...	
block5a_bn (BatchNormalizatio...	(None, 8, 8, 624)	2,496	block5a_dwconv2[...	
block5a_activation (Activation)	(None, 8, 8, 624)	0	block5a_bn[0][0]	
block5a_se_squeeze (GlobalAveragePool...	(None, 624)	0	block5a_activati...	
block5a_se_reshape (Reshape)	(None, 1, 1, 624)	0	block5a_se_squee...	
block5a_se_reduce	(None, 1, 1, 26)	16,250	block5a_se_resha...	