Garbage Classification with EfficientNetV2B2

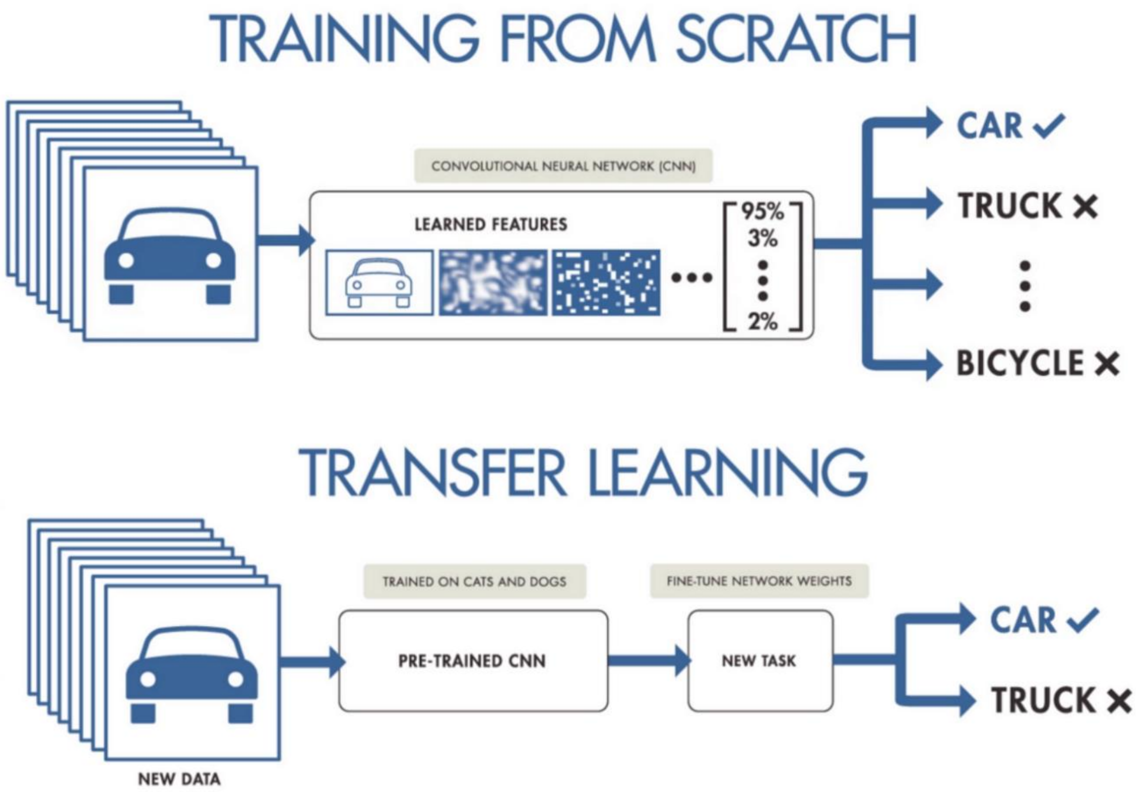
Project Description

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 specific task is reused as the starting point for a model on a different but related task. It also allows us to build accurate models in a time-saving way by starting from patterns learned when solving a different problem. This approach is beneficial when there is limited data for the new task, as the pre-trained model already has learned features that can be adapted. Transfer learning can significantly improve models' performance and efficiency in domains like computer vision and natural language processing.

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.

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.

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:

These computed weights are then passed to the model.

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)

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.

4. Model Training

- Build the model architecture using Sequential or Functional API.

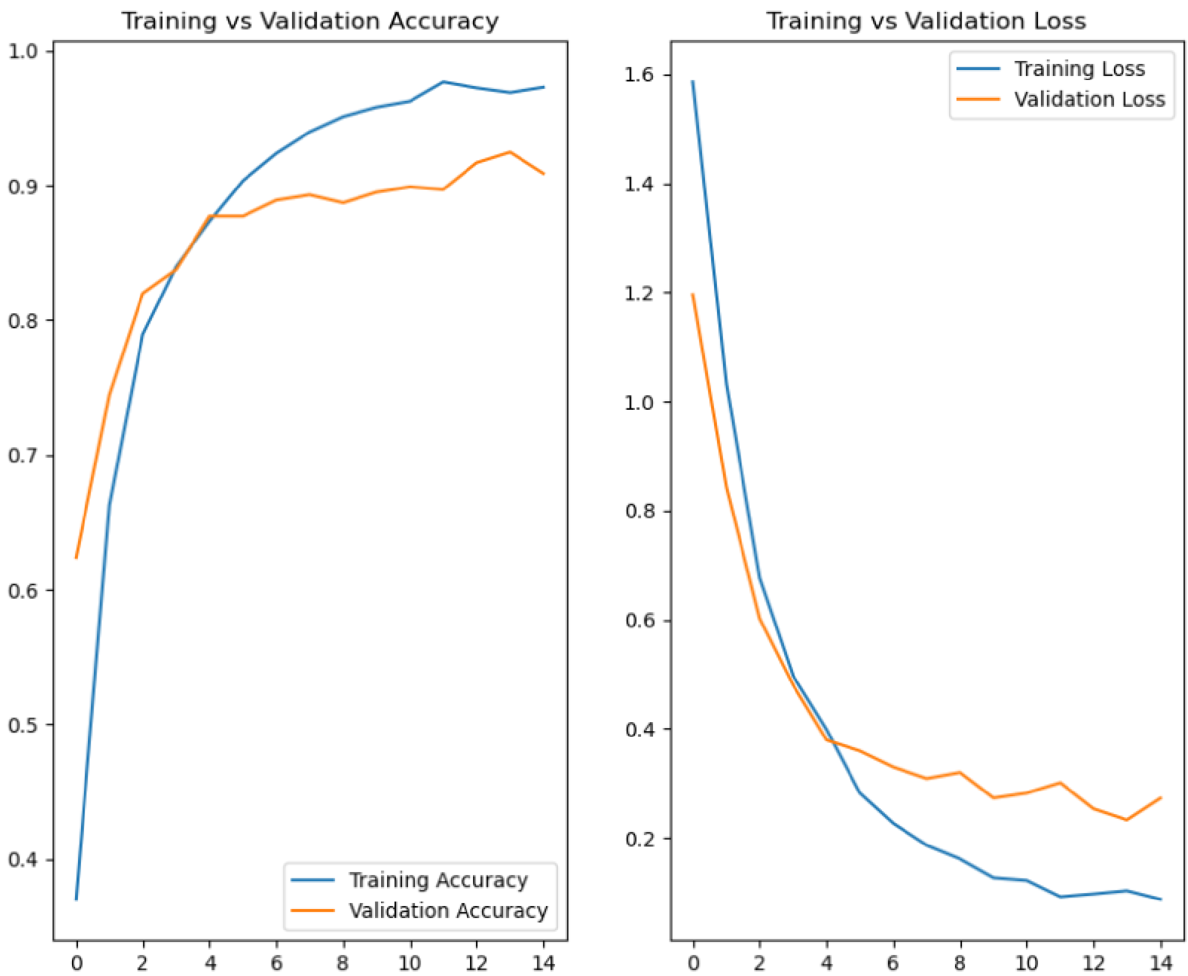
- Compile the model with loss function (sparse_categorical_crossentropy), optimizer (e.g., Adam), and evaluation metrics (accuracy).

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.

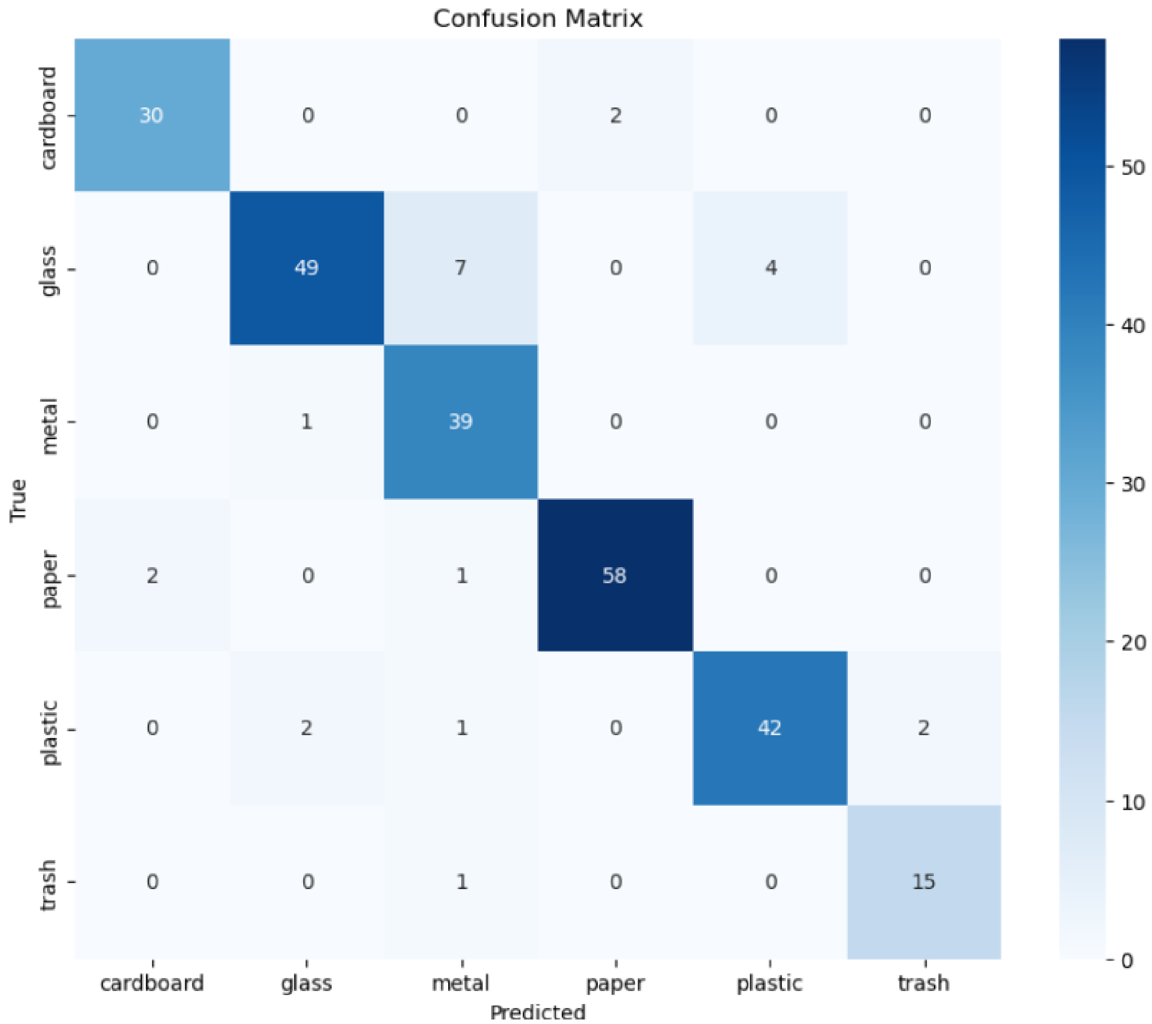
Model Architecture and Layer Utilities

- **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



6. Model Evaluation

- Plot training and validation accuracy/loss curves.
- Evaluate model performance on validation or test set.
- Use metrics like:
    - **Confusion Matrix**
    - **Classification Report** (Precision, Recall, F1-score)
    - confusion_matrix, classification_report: To evaluate the model's classification performance.



7. Final Testing and Save the Model

- Evaluate the final model on the unseen **test dataset**

🌐 **Gradio Interface and Preprocessing**

- gr: To build a web interface for the model.
- PIL.Image: For handling image input in Gradio.
- preprocess_input: Preprocessing method for EfficientNet.
- load_model: For loading a saved model for inference.

Conclusion

- The image classification model demonstrates strong accuracy in identifying objects, leveraging deep learning to refine predictions effectively. Its robust performance ensures reliable classification, making it a valuable tool for various applications