Deep Learning for Waste Recognition: An Image Classification System

Applying Deep Neural Networks to
Automate the Sorting of Waste Materials.

Rubèn Lallave Fabregat

The Waste Management Challenge

INDUSTRIAL LEVEL:

- Inefficiency of Manual Sorting (slow and high manpower costs)
- Human Error and Contamination

GENERAL LEVEL:

- Environmental Impact (less recycling, more landfills)
- Difficulties for Users in Household Sorting:
  - Confusion about what goes in each bin
  - Lack of knowledge about the recyclability of certain materials
  - Apathy or lack of motivation

Intelligent Waste Sorting with Deep Learning

Developing an Automated Image Recognition System for Efficient Categorization of Recyclable Waste.

Leveraging Deep Learning:

Deep Learning, through Convolutional Neural Networks (CNNs), enables the automatic learning of complex visual features directly from images.

Image-Based Classification:

Identify and classify materials such as paper, plastic and glass, also, reduces reliance on human knowledge at the point of classification.

Automation of the Sorting Process

Potential for for integration into existing waste management systems (conveyor belts, sorting robots). Automation can significantly increase the speed and efficiency of the waste sorting process.

Potential for User Information (Future Vision):

Help end-users to identify and correctly sort waste in their homes through Apps.

Data Collection:

Step 1: Using dataset Garbage Classification (12 classes) ✅

Step 2: Getting more images on Google Images (Web Scrapping) ❌

Step 3: Getting images from API's Pexels and Pixabay ✅ ❌

Step 4: Unifying with dataset Garbage Classification (6 classes) ✅

Method Selection

Learning from Scratch or Transfer Learning (Pre-trained Models)?

CNNs are the primary architecture for automatically extracting spatial features via convolutional layers, making them fundamental to many high-performing pre-trained models.

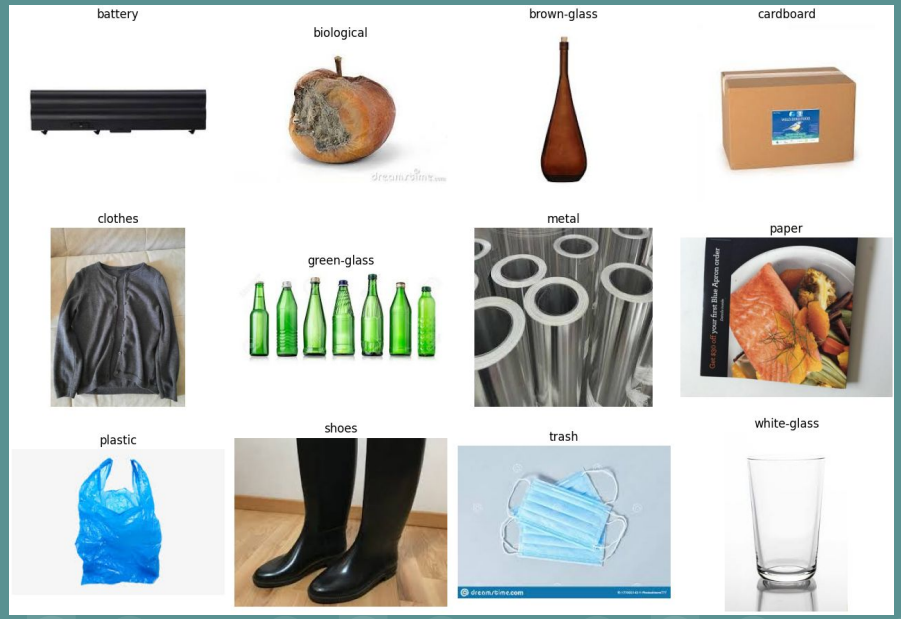
And now, which pre-trained model should we select? VGG, ResNet, Inception, EfficientNet...?

Leveraging ResNet50: Our Feature Extraction Powerhouse

ResNet50 is a deep and effective Convolutional Neural Network (CNN) for image recognition. Its key innovation, "residual connections," enables training significantly deeper networks.

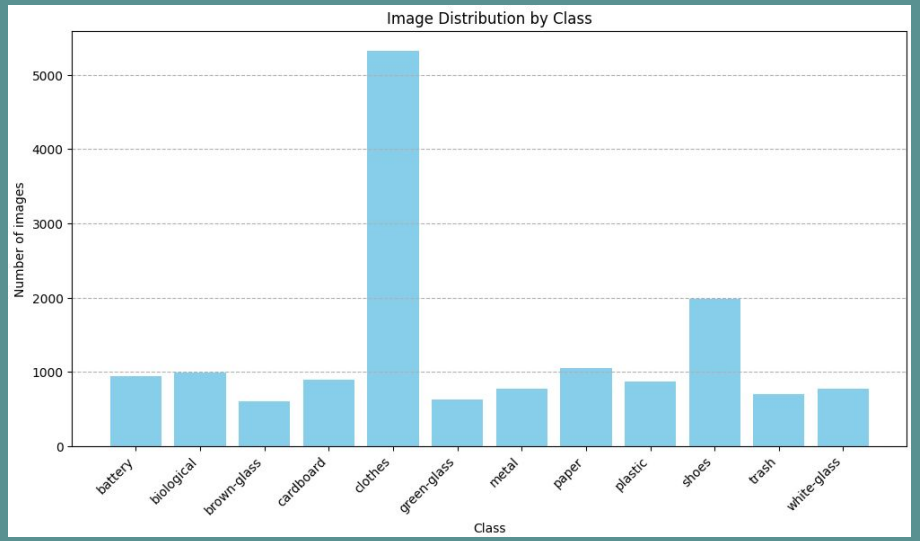
- ResNet architectures consistently achieve top results in image classification benchmarks, showcasing their high performance.
- ResNet50's deep layers and residual connections enable **Robust Feature Extraction**, learning intricate image features essential for distinguishing waste types.
- Pre-trained ResNet50 on ImageNet provides a significant **Transfer Learning Advantage** with rich visual features, reducing our data and training needs and allowing focus on adapting to our waste categories.

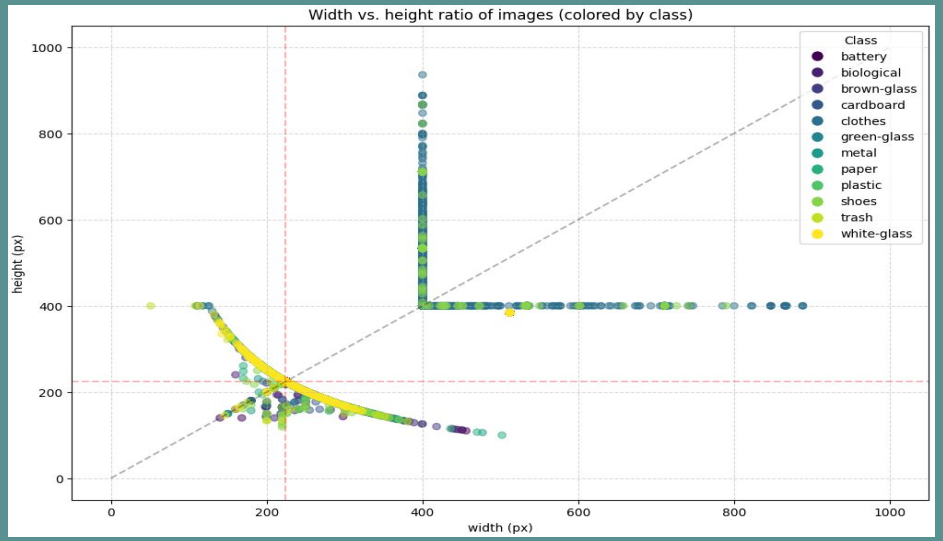
Exploring dataset & getting visualizations

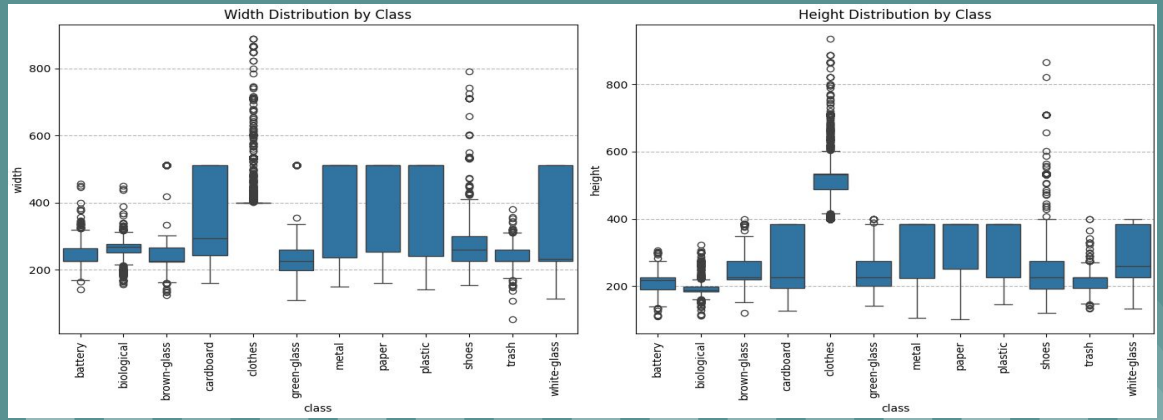


os.listdir(dataset_path)	Asking that tool to give you a list of all the “names” inside the junk folder we saved in dataset_path.
Image.open(img_path)	This uses a tool called Image (from a library called Pillow or PIL) to open the image
mpimg.imread(...)	takes the address of the image file and reads it, converting it directly into a “numeric array”.
io.imread(...)	reads the image file and converts it directly into a NumPy array
color.rgb2gray(img)	takes that color image and converts it to a grayscale image

Exploratory Data Analysis







Exploring dataset & getting visualizations

Dividing the Dataset: We split the image data into training, validation, and test sets. The validation set was crucial for monitoring performance during training to optimize the model. The final test set served for unbiased evaluation.

Resizing Images: All images were resized to a standard 224x224 pixels, the expected input size for the pre-trained ResNet50 model, ensuring consistency and efficiency.

Model Construction: We loaded the pre-trained ResNet50 base model, excluding its classification layers (include_top=False) and freezing its weights to preserve learned features. We then added custom layers on top – pooling, dense, batch normalization, and dropout – tailored for our waste classification. Finally, we compiled the model with Adam optimizer and categorical cross-entropy loss.

```
 Image.open(img_path) as img:
img = img.convert('RGB')  # Consistent Image Format
img_resized = resize(np.array(img), IMG_SIZE, anti_aliasing=True).a
print(f"Range after resize for {img_path}: Min={img_resized.min()},
images.append(img_resized)
labels.append(class_to_index[class_name])
```

```
# Freeze base model layers to prevent training
for layer in base_model.layers:
    layer.trainable = False

# Add custom layers on top
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(500, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
predictions = Dense(len(classes), activation='softmax')(x)
```

Training Configuration & Optimization

To effectively train our Deep Learning model for waste classification, we implemented key configurations and optimization strategies. These choices aimed for optimal performance, overfitting prevention, and efficient training. We used model-specific preprocessing for ResNet50 compatibility, callbacks to dynamically adjust learning and prevent overfitting, and balanced training parameters for computational efficiency and stability.

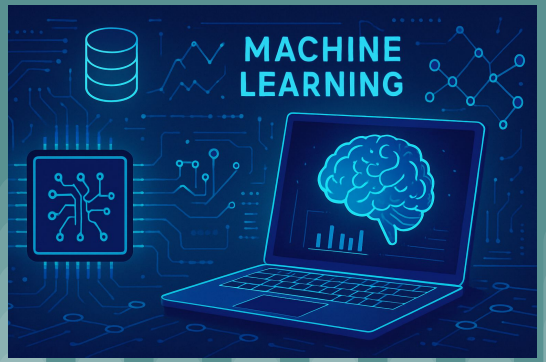
`preprocess_input(X)`: Normalized images as expected by pre-trained ResNet50 for optimal performance.

`EarlyStopping`: Halts training if validation loss doesn't improve for 10 epochs, restoring best weights.

`ReduceLROnPlateau`: Reduces learning rate by 0.2 if validation loss plateaus for 5 epochs.

`epochs = 50`: Maximum 50 training cycles (Early Stopping may end sooner).

`batch_size = 32`: Balances computational efficiency and learning stability.



Initial results:

Our initial training, likely yielded suboptimal performance.

We observed potential overfitting and limited generalization, possibly due to the small dataset size and lack of variability during training.

Evaluation metrics on the validation set indicated room for significant improvement in accuracy and loss.

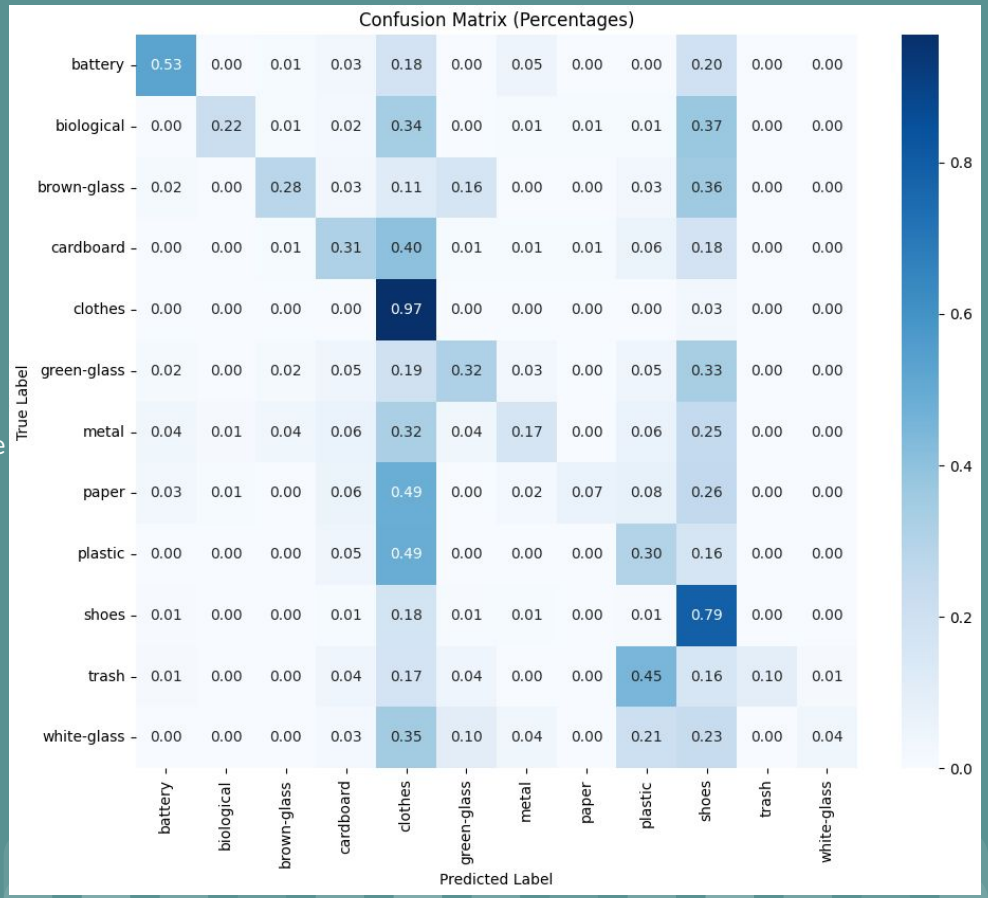
Model Improvement:

Focused Classification: We streamlined our classification task by removing clothes and shoes, high-volume classes that were skewing the model.

Data Augmentation & Efficient Handling (ImageDataGenerator): We used ImageDataGenerator with flow_from_dataframe for real-time data augmentation (rotations, etc.) to improve generalization and efficiently handle large datasets directly from disk, avoiding RAM issues.

Normalization: Images were rescaled to the 0-1 range (`rescale=1./255`) for better gradient flow and model stability.

ModelCheckpointing: We implemented saving the best model weights based on validation loss, ensuring we retain the model with the highest generalization capability.



Model Enhancements:

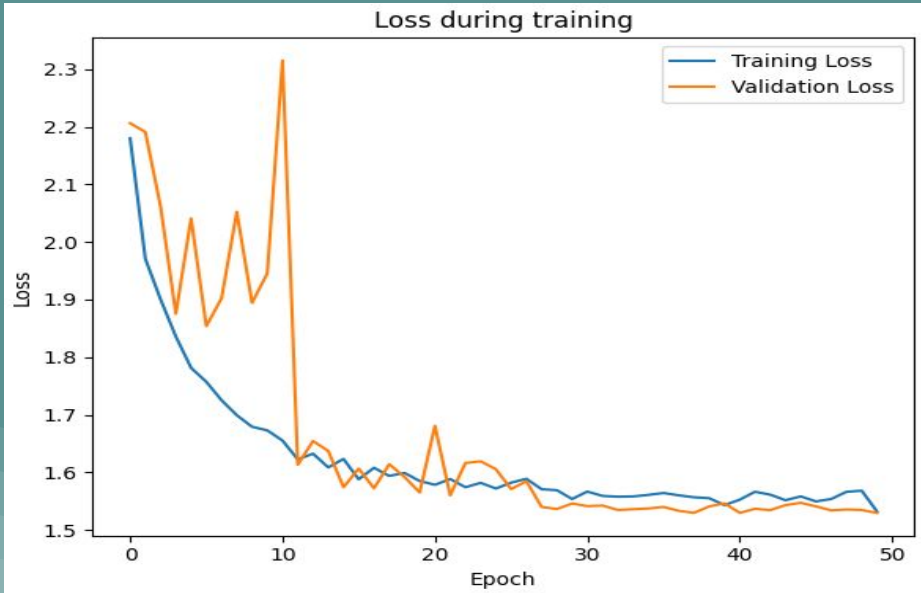
Notably lower validation loss indicates better generalization and fewer errors on new data.

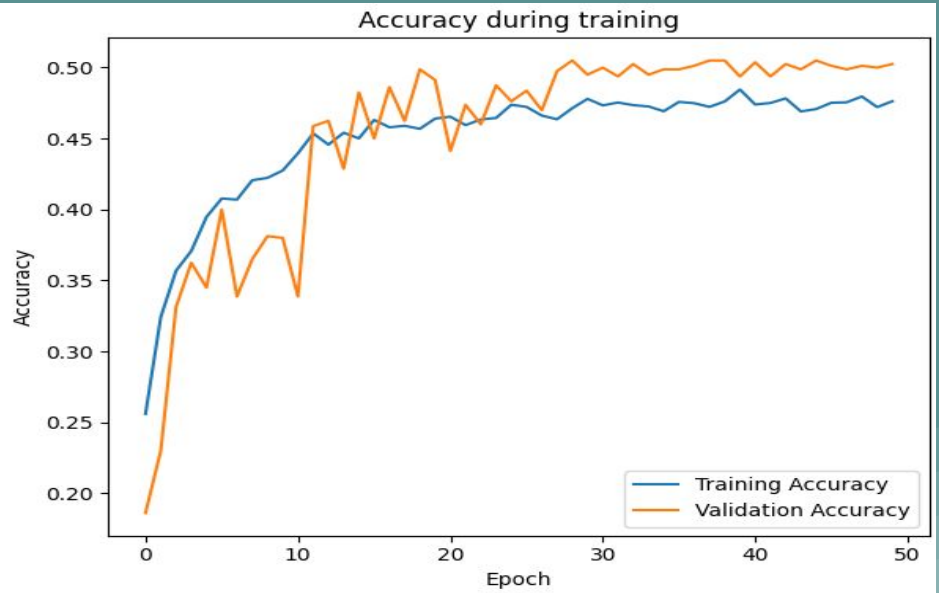
Enhanced model shows significantly higher accuracy on new waste images.

Smaller training-validation gap indicates less overfitting and better generalization.

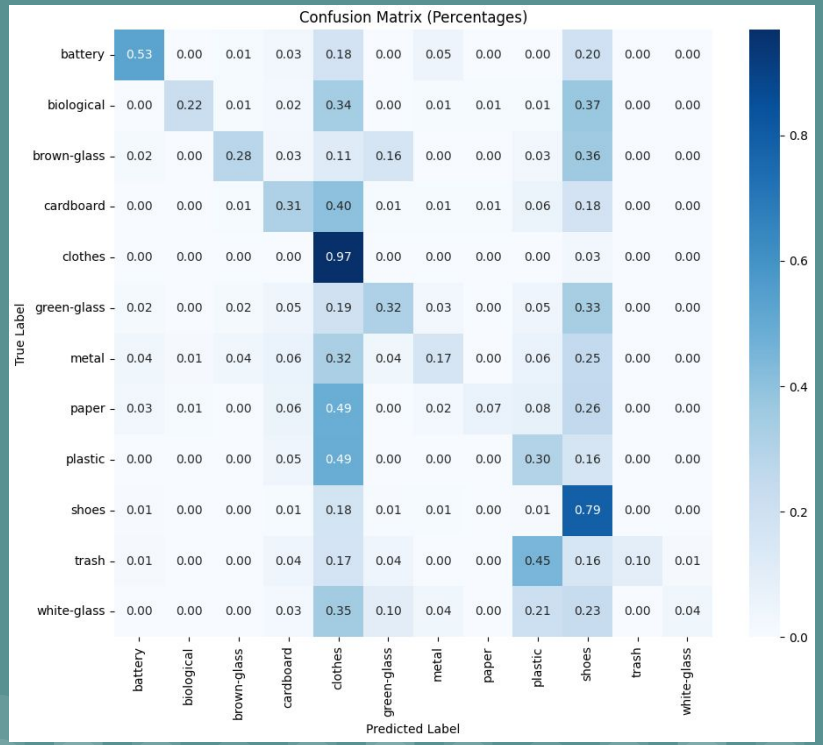
Stable validation performance suggests a robust training process.

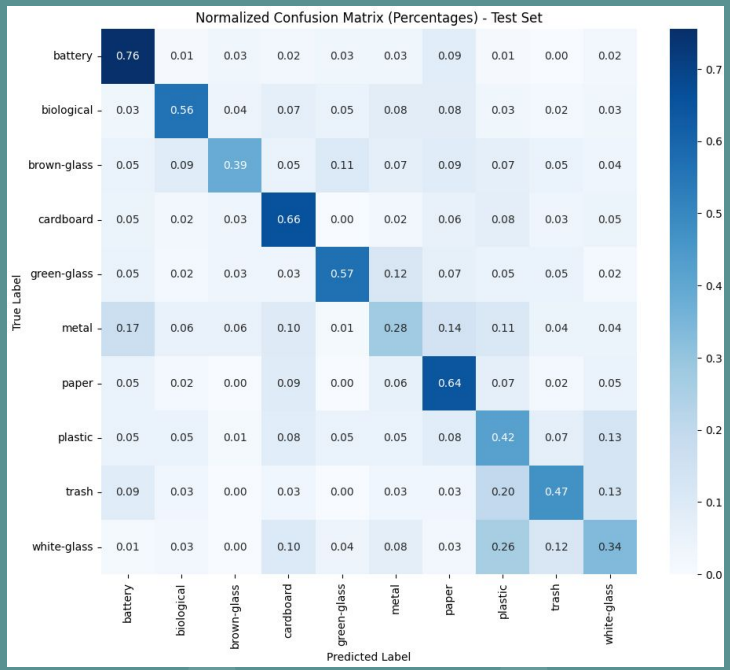
The model learned efficiently, reaching higher performance.





Significant Performance Boost After Optimizations





Train Loss: 1.4225
Train Accuracy: 0.5351

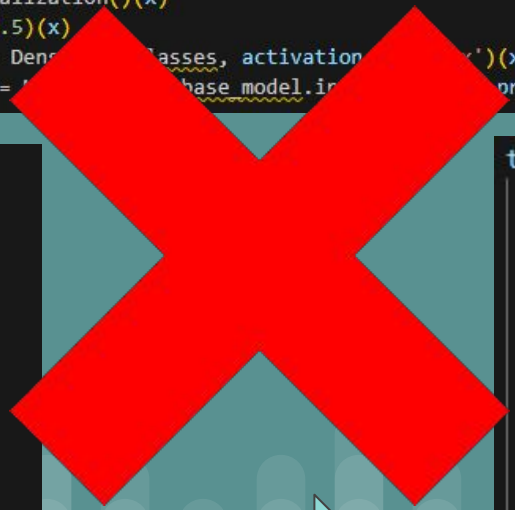
Validation Loss: 1.5295
Validation Accuracy: 0.5025

Test Loss: 1.5209
Test Accuracy: 0.5238

Exploring Model Optimization: Unsuccessful Attempts

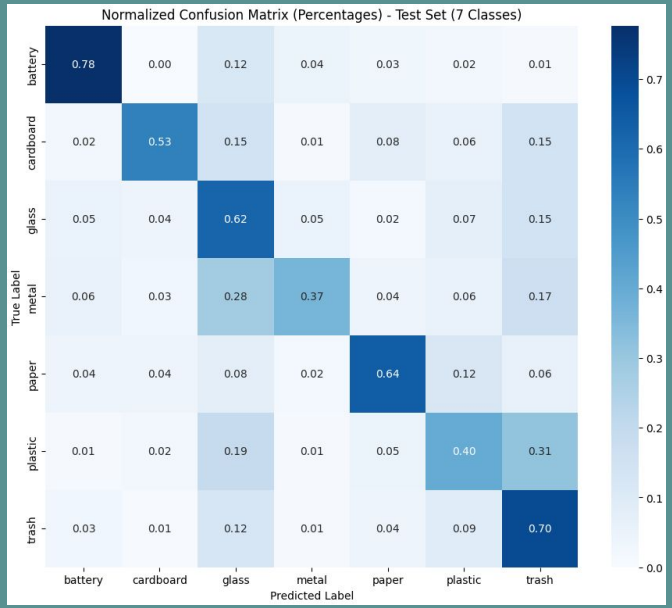
```python
# Freeze almost all layers
for layer in base_model.layers[:-20]:
    layer.trainable = False

# Adding custom layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
predictions = Dens        lasses, activation        ')(x)
model_frozen =          base_model.in          predictions)
```

```python
# Data generators with the new classes
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest',
    preprocessing_function=preprocess_input
)
```



```python
train_datagen_augmented = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    preprocessing_function=preprocess_input
)
```

Refined Classification: Reducing to 7 Categories



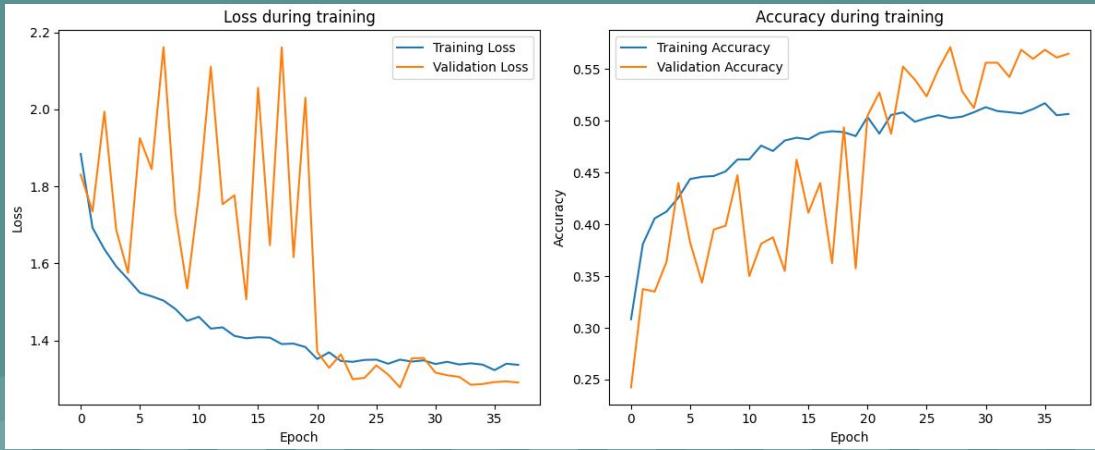
Train Loss: 1.2326
Train Accuracy: 0.5620

Validation Loss: 1.2783
Validation Accuracy: 0.5713

Test Loss: 1.1641
Test Accuracy: 0.6012

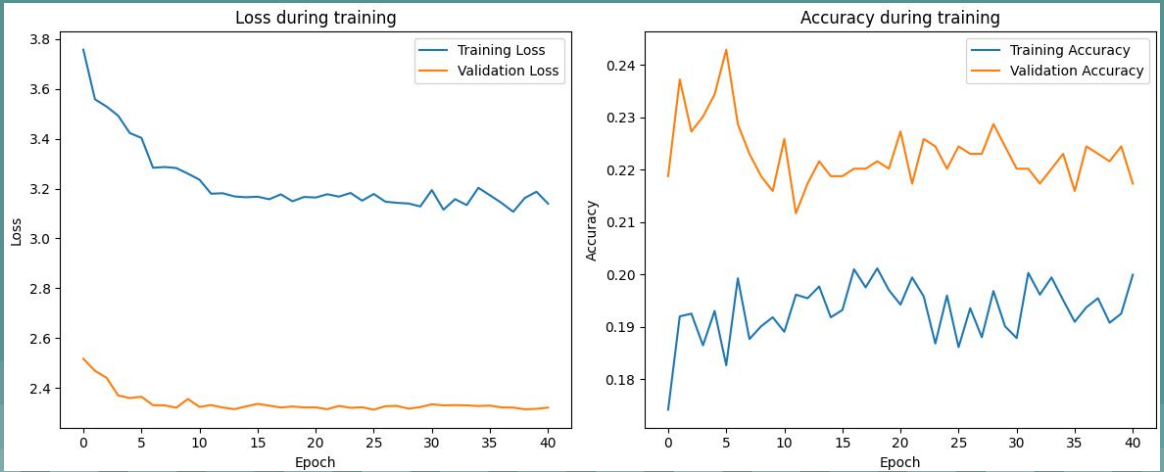
Reverting to a ResNet-based architecture, similar to our initial success, we refined the output to 7 categories. This involved mapping "biological" waste into "trash" and consolidating the different "glass" types into a single "glass" category. This focused approach aimed to improve classification accuracy for our primary interest areas.

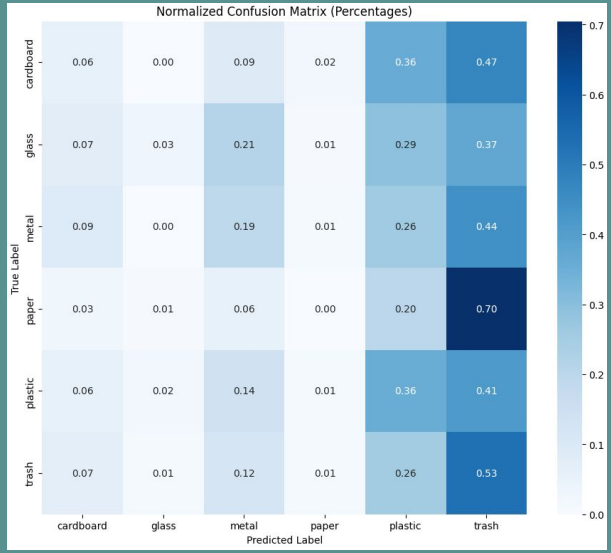
```
CLASS_MAPPING = {
    'battery': 'battery',
    'biological': 'trash',
    'brown-glass': 'glass',
    'cardboard': 'cardboard',
    'green-glass': 'glass',
    'metal': 'metal',
    'paper': 'paper',
    'plastic': 'plastic',
    'trash': 'trash',
    'white-glass': 'glass'
}
```



Explored Alternative Architectures (EfficientNetB0)

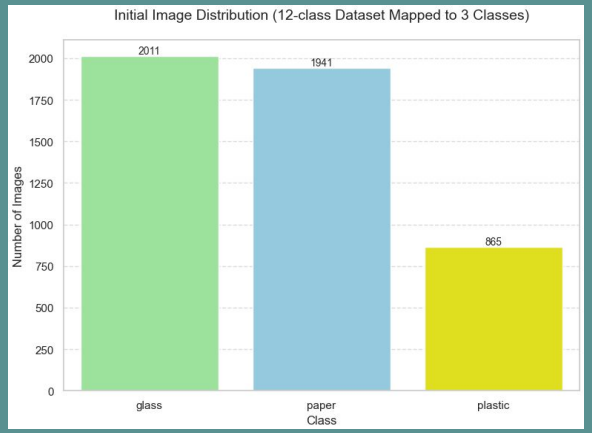
Recognizing the potential for further performance gains, I also experimented with the EfficientNetB0 architecture. I hypothesized that its efficient scaling strategy could offer a better balance between model size, computational cost, and accuracy for our specific waste classification task.

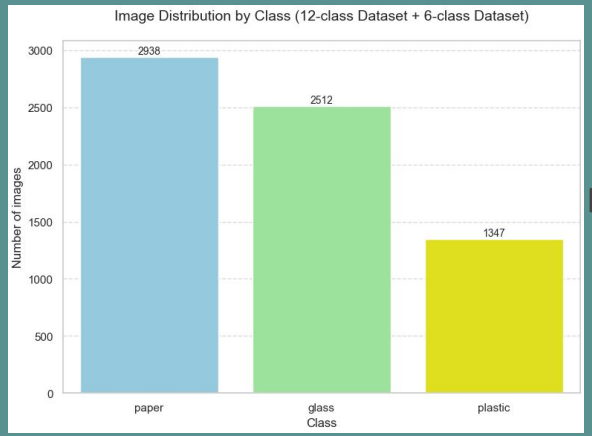


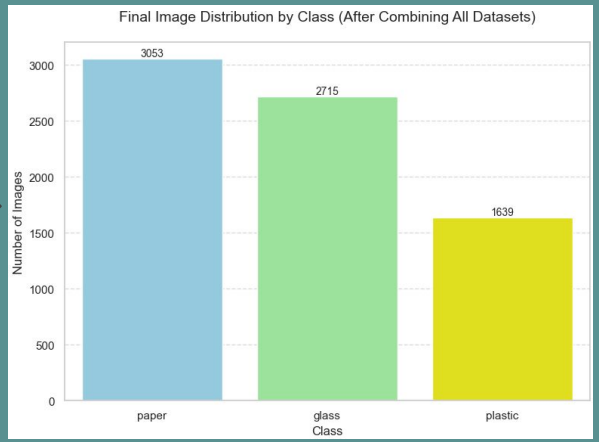


EfficientNetB0 resulted in a notable decrease in classification accuracy and poor generalization, with significant misclassifications observed.

Addressing Class Imbalance: Integrating External Data



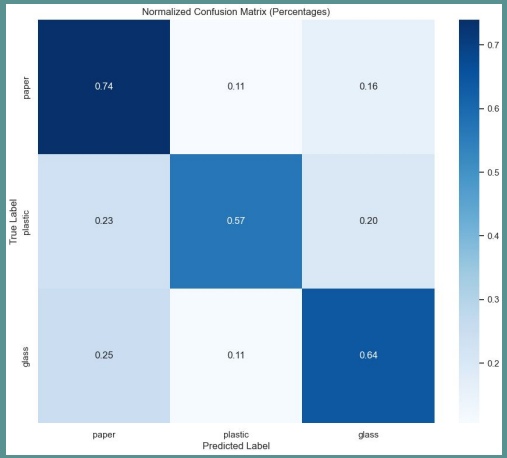


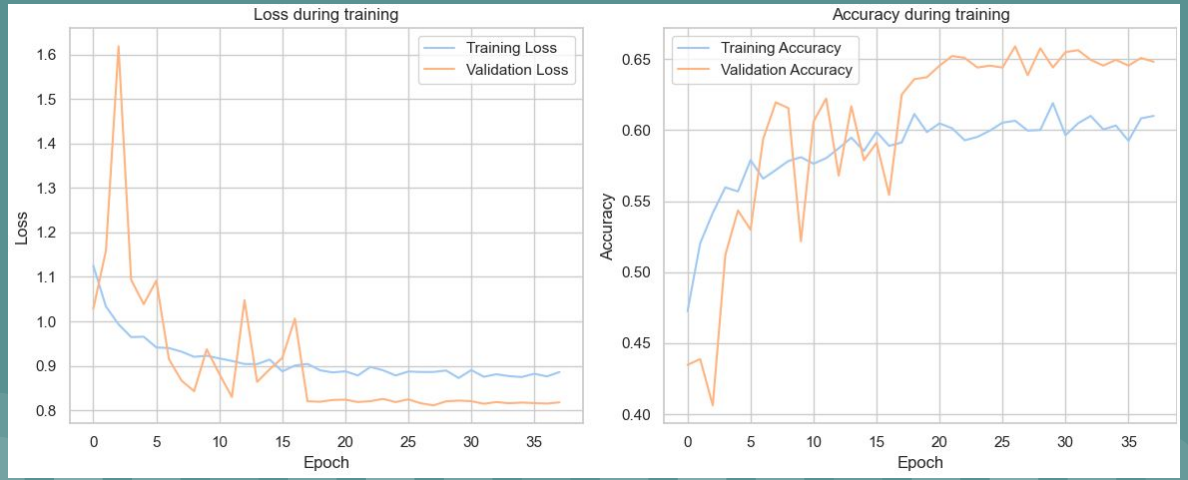


To mitigate potential class imbalance and bolster the representation of underrepresented categories, I strategically augmented our initial dataset. This involved programmatically acquiring additional images via APIs and targeted Google Image searches for specific waste types. The goal was to create a more balanced dataset, ensuring the model receives sufficient examples for each class to learn effectively and avoid bias towards dominant categories.

Effective 3-Class Waste Classification: Training Performance

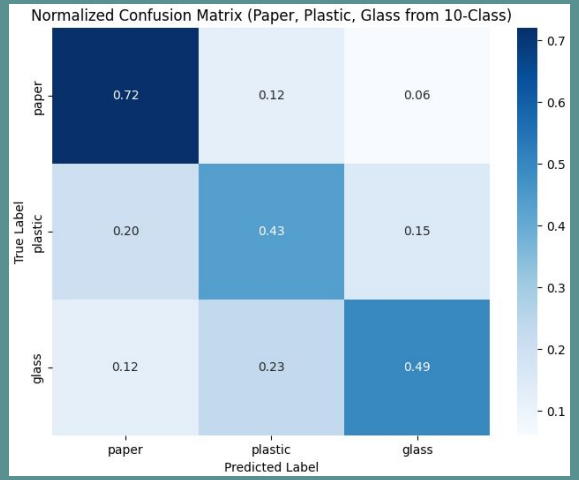
The provided training graphs illustrate successful learning for the 3-class waste classification problem (paper, plastic, glass). The **Loss during training** plot shows a clear and consistent decrease in both training and validation loss over the epochs. The stabilization of the validation loss near the training loss indicates good generalization and minimal overfitting.



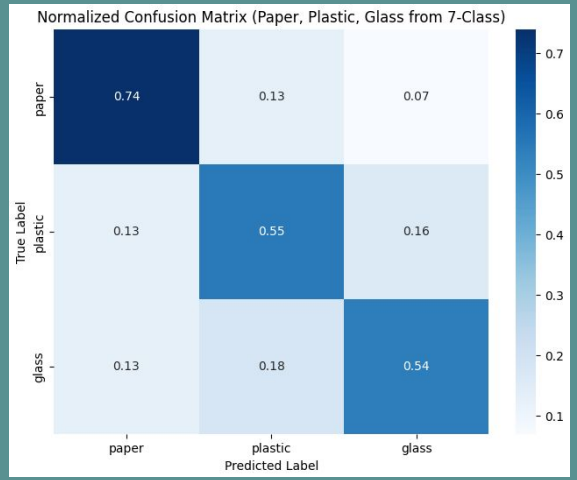


The **Accuracy during training** plot shows a significant rise in both training and validation accuracy, plateauing around 65% for validation. This indicates effective learning to distinguish the 3 waste categories, with close tracking between curves suggesting good generalization on unseen data, representing a focused improvement on core recyclables.

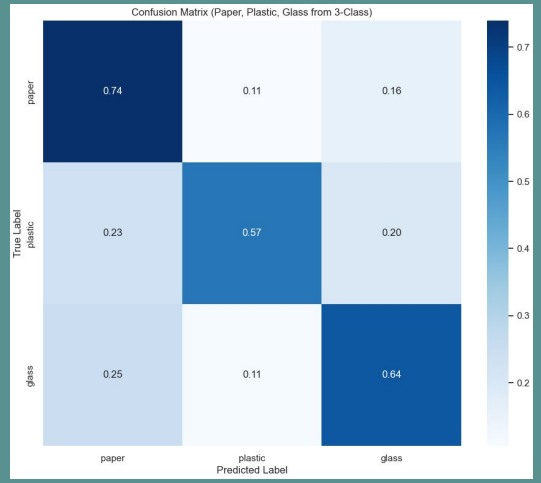
Progressive Improvement in 3-Class Classification with Data Integration



Test Loss: 1.5209
Test Accuracy: 0.5238



Test Loss: 1.1641
Test Accuracy: 0.6012

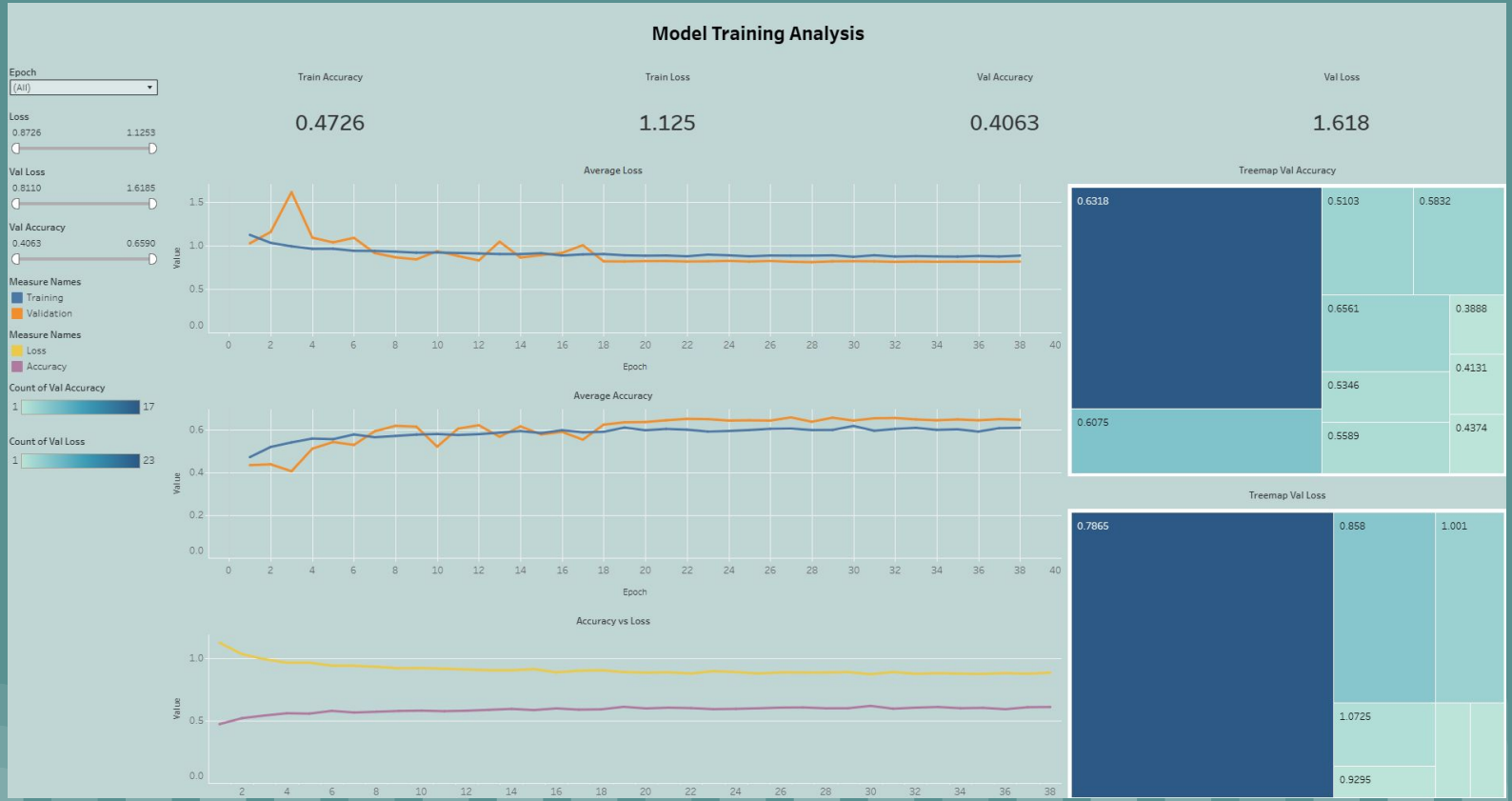


Test Loss: 0.8072
Test Accuracy: 0.6576

Focusing on 3 core classes (paper, plastic, glass) with integrated data significantly boosted test accuracy (0.53 to 0.66) and reduced loss (1.52 to 0.81).

While 'paper' and 'glass' classification improved, 'plastic' variability suggests a need for further targeted enhancements.

TABLEAU


Model Training Analysis
Epoch
(All)
Loss
0.8726
1.1253
Val Loss
0.8110
1.6185
Val Accuracy
0.4063
0.6590
Measure Names
Training
Validation
Measure Names
Loss
Accuracy
Count of Val Accuracy
1
17
Count of Val Loss
1
23
Train Accuracy
0.4726
Train Loss
1.125
Val Accuracy
0.4063
Val Loss
1.618
Average Loss
Value
Epoch
Average Accuracy
Value
Epoch
Accuracy vs Loss
Value
Treemap Val Accuracy
0.6318
0.5103
0.5832
0.6561
0.3888
0.4131
0.5346
0.6075
0.5589
0.4374
Treemap Val Loss
0.7865
0.858
1.001
1.0725
0.9295

Project Conclusions: Towards Effective Waste Classification

Iterative refinement, focusing on 3 classes (paper, plastic, glass) from a more granular approach, improved accuracy.

Focusing on core recyclables (paper, plastic, glass) significantly **improved training stability and classification performance** (accuracy/loss).

External data integration enabled higher generalization and a validation accuracy of approximately 66% in the 3-class task.

To further improve, **future work** will focus on enhancing 'plastic' classification accuracy through **more** specific **data** or **model changes**.

Project Challenges and Lessons Learned

Long Training Times & Interruptions: Training was computationally intensive and prone to interruptions due to system restarts before implementing proper ModelCheckpoint.

Limited Impact of Model Enhancements: Attempts to improve performance using EfficientNetB0 and techniques like class balancing/layer freezing yielded no significant gains.

Difficulty of Granular Classification: Accurately classifying 10/12 visually similar waste categories proved challenging, leading to lower initial accuracy.

Data Variability and Ambiguity: The diverse and often ambiguous appearance of waste items (shape, contamination, lighting) made robust classification difficult.

Initial Class Imbalance: Uneven distribution of data across the more granular classes likely hindered the model's ability to learn effectively for all categories initially.

This project was made by Rubèn Lallave Fabregat

Thank you!

Danke!

¡Gracias!

Obrigado!

Gràcies!

Dziękuję!

Sharing and working with all of you during this bootcamp has been wonderful. It's been a pleasure to present my final project to you.