EN3150 Assignment 03

Simple Convolutional Neural Network
to Perform Classification

Submitted by:

220491B Praveen V.V.J
220399B Muftee M M M
200389U Mathujan S
220502M Rajinthan R

Department of Electronic and Telecommunication Engineering

Instructor:

Sampath K. Perera

November 1, 2025

Contents

1 Environment and Dataset

1.1 Environment Setup

This assignment was completed using the **Python** programming language. The core libraries used for building and training the neural network are **TensorFlow** and its high-level API, **Keras**.

Key Libraries:

- **Numpy:** For numerical operations and array manipulation
- **Matplotlib & Seaborn:** For data visualization (loss plots, confusion matrix)
- **PIL (Pillow):** For loading and processing images from the dataset
- **Scikit-learn:** For splitting the dataset and calculating evaluation metrics

1.2 Dataset Preparation

The dataset chosen is the **RealWaste dataset**, sourced from Kaggle. This dataset contains images of waste categorized into multiple classes.

Preprocessing Steps:

- **Loading:** Images were loaded from folders, with each folder name representing a class
- **Resizing:** All images were resized to a uniform shape of **128 × 128** pixels
- **Color Channels:** Images were converted to **RGB** (3 color channels)
- **Normalization:** Pixel values were scaled from [0, 255] to [0.0, 1.0] by dividing by 255.0

2 Dataset Split

The dataset was divided into three distinct subsets using Scikit-learn's `train_test_split` function, adhering to a 70%-15%-15% ratio. A stratified split was used to ensure class distribution was preserved in all three sets.

Split Procedure:

1. First split: 70% **Training Set** and 30% temporary set
2. Second split: The 30% set was divided equally into 15% **Validation Set** and 15% **Testing Set**

3 CNN Model Architecture

(Corresponds to Q4) A Sequential Keras model was constructed. The architecture from the provided code, which uses 'padding='same'', is as follows. Note: 'padding='same'' means the spatial dimensions (height/width) do not shrink after the convolution.

Table 1: Custom CNN Architecture

Layer Type	Configuration	Output Shape (after layer)
Input Layer	(128, 128, 3)	(128, 128, 3)
Conv2D (1)	32 filters, 3×3, ReLU, padding='same'	(128, 128, 32)
MaxPooling2D (1)	2×2 pool size	(64, 64, 32)
Conv2D (2)	64 filters, 3×3, ReLU, padding='same'	(64, 64, 64)
MaxPooling2D (2)	2×2 pool size	(32, 32, 64)
Conv2D (3)	128 filters, 3×3, ReLU, padding='same'	(32, 32, 128)
MaxPooling2D (3)	2×2 pool size	(16, 16, 128)
Conv2D (4)	256 filters, 3×3, ReLU, padding='same'	(16, 16, 256)
MaxPooling2D (4)	2×2 pool size	(8, 8, 256)
Conv2D (5)	512 filters, 3×3, ReLU, padding='same'	(8, 8, 512)
MaxPooling2D (5)	2×2 pool size	(4, 4, 512)
Flatten	—	(8192)
Dense (FC 1)	512 units, ReLU, L2 reg. (0.001)	(512)
Dropout	Rate: 0.2 (20%)	(512)
Dense (Output)	9 units, Softmax	(9)

4 Network Parameters

(Corresponds to Q5) The network parameters were carefully chosen based on best practices for image classification:

Convolutional Layers:

- **Filter Sizes:** Progressive increase ($32 \rightarrow 64 \rightarrow 128 \rightarrow 256 \rightarrow 512$) allows the network to learn simple features in early layers and combine them into complex features in deeper layers
- **Kernel Size:** 3×3 for all convolutional layers—a standard, efficient size that captures local spatial patterns

Fully Connected Layers:

- **Dense Layer Size:** 512 units to learn complex combinations of high-level features
- **Dropout Rate:** 0.2 (20%) provides regularization without hindering learning
- **L2 Regularization:** Weight decay of 0.001 penalizes large weights to reduce overfitting

5 Justification for Activation Functions

(Corresponds to Q6)

5.1 ReLU (Rectified Linear Unit)

ReLU, defined as $f(x) = \max(0, x)$, was chosen for all convolutional and hidden dense layers for three main reasons:

1. **Computational Efficiency:** Simple threshold operation at zero, much faster than sigmoid or tanh
2. **Non-Saturating Gradient:** For positive inputs, gradient is always 1, preventing the vanishing gradient problem

3. **Sparsity:** Sets negative inputs to zero, creating sparse representations that improve network efficiency

5.2 Softmax

The Softmax function was used for the output layer, which is standard for multi-class classification:

- **Probability Distribution:** Converts raw logits into a probability distribution
- **Interpretability:** Each output value is between 0 and 1, with all outputs summing to 1, representing the model's confidence for each class

6 Model Training

(Corresponds to Q7) The model was trained for **20 epochs** with a batch size of 32. The `ModelCheckpoint` callback monitored `val_loss` and saved only the "best" model, preventing overfitting and ensuring evaluation on the model that generalized best to validation data.

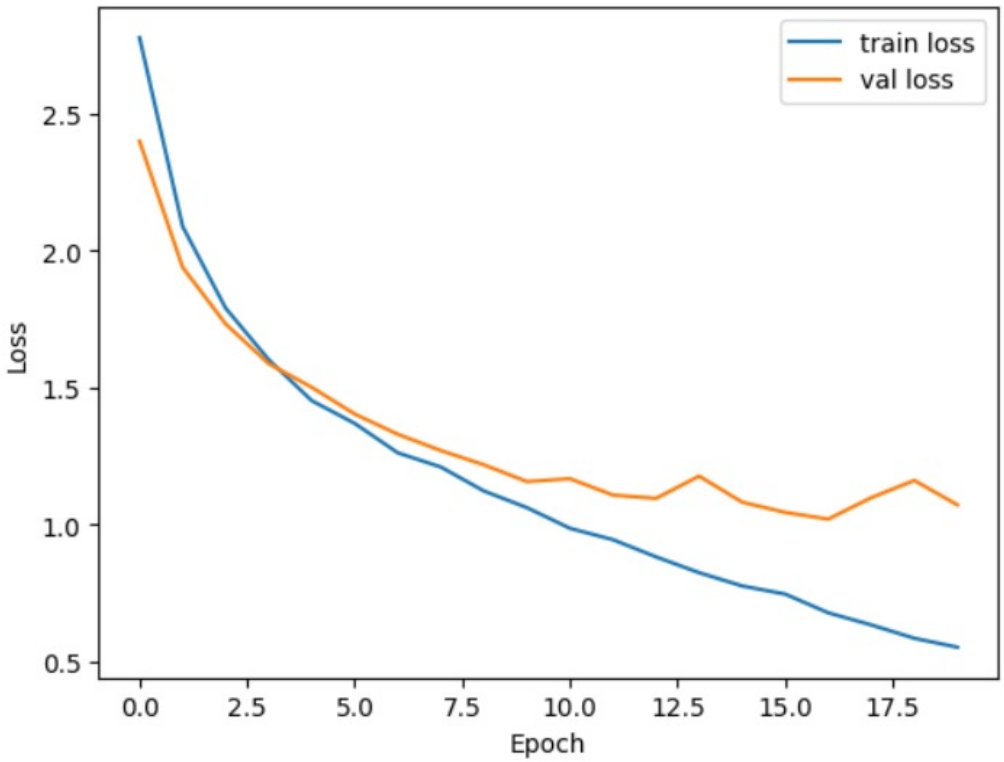


Figure 1: Custom CNN (Adam Opt.): Training and Validation Loss vs. Epochs

7 Optimizer

(Corresponds to Q8) The primary optimizer chosen was **Adam (Adaptive Moment Estimation)**.

Reasoning: Adam is highly effective and popular, combining advantages of two other optimizers:

- **RMSprop:** Maintains adaptive learning rates for each parameter based on recent squared gradients

- **Momentum:** Accelerates convergence by adding a fraction of the previous update, dampening oscillations

Adam is generally fast, efficient, and requires minimal manual tuning of learning rates, making it an excellent default choice.

8 Learning Rate Selection

(Corresponds to Q9) The learning rate was set to **1e-4 (0.0001)** for the Adam optimizer and **0.01** for the SGD experiments.

For Adam, 1e-4 is a common, conservative starting point that ensures stable convergence. For SGD, 0.01 is a traditional starting point, which was kept constant for both standard SGD and SGD with Momentum to allow for a fair comparison of the momentum parameter's impact.

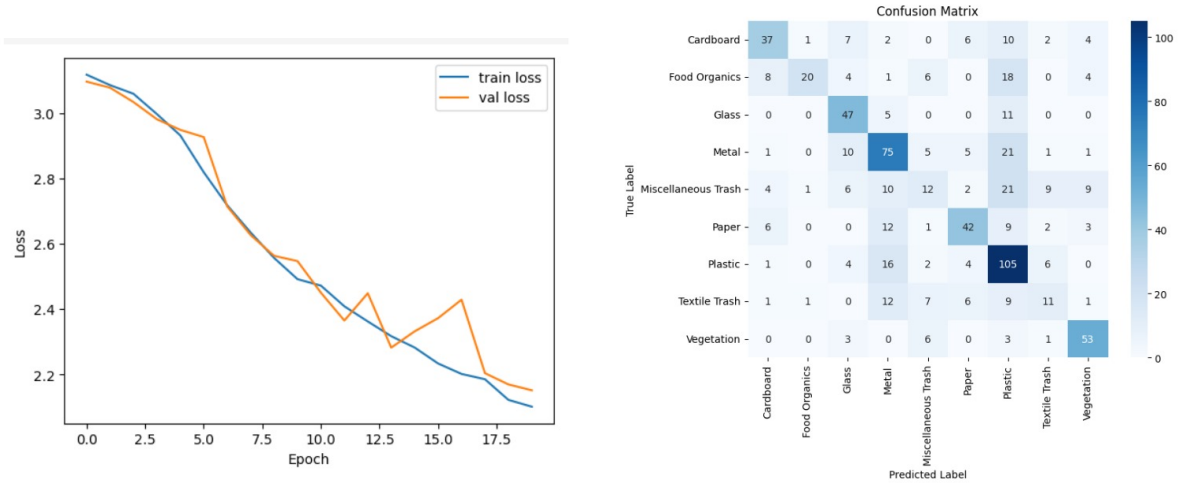
9 Optimizer Performance Comparison

(Corresponds to Q10) To evaluate the chosen Adam optimizer, its performance was compared against (a) standard Stochastic Gradient Descent (SGD) and (b) SGD with Momentum. The same custom CNN architecture was used for all three experiments.

9.1 Comparison (a): Standard SGD

The model was trained using `optimizer=SGD(learning_rate=0.01)`.

- **Test Loss:** 2.0864
- **Test Accuracy:** 0.5638 (56.38%)



(a) SGD: Training/Validation Loss

(b) SGD: Confusion Matrix

Figure 2: Performance of Custom CNN with Standard SGD Optimizer

The classification report (Table 2) shows poor performance across most classes, particularly 'Miscellaneous Trash' (F1: 0.21) and 'Textile Trash' (F1: 0.28).

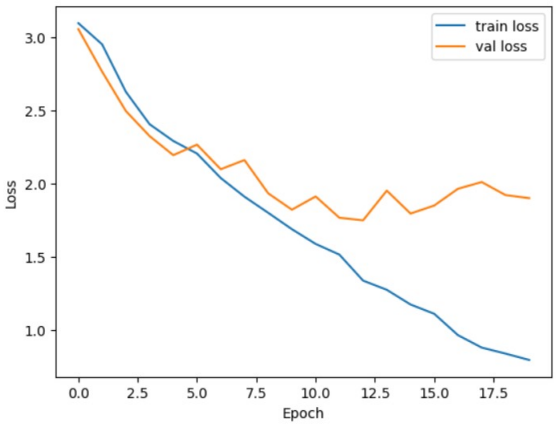
Table 2: Classification Report: Standard SGD

Class	Precision	Recall	F1-Score	Support
Cardboard	0.6379	0.5362	0.5827	69
Food Organics	0.8696	0.3279	0.4762	61
Glass	0.5802	0.7460	0.6528	63
Metal	0.5639	0.6303	0.5952	119
Miscellaneous Trash	0.3077	0.1622	0.2124	74
Paper	0.6462	0.5600	0.6000	75
Plastic	0.5072	0.7609	0.6087	138
Textile Trash	0.3438	0.2292	0.2750	48
Vegetation	0.7067	0.8030	0.7518	66
Accuracy			**0.5638**	**713**
Macro Avg	**0.5737**	**0.5284**	**0.5283**	**713**
Weighted Avg	**0.5682**	**0.5638**	**0.5452**	**713**

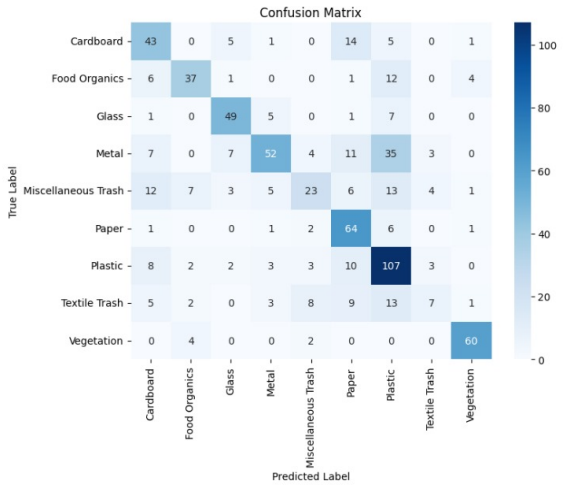
9.2 Comparison (b): SGD with Momentum

The model was trained using `optimizer=SGD(learning_rate=0.01, momentum=0.9)`.

- **Test Loss:** 1.8313
- **Test Accuracy:** 0.6199 (61.99%)



(a) SGDM: Training/Validation Loss



(b) SGDM: Confusion Matrix

Figure 3: Performance of Custom CNN with SGD + Momentum Optimizer

The classification report (Table 3) shows a noticeable improvement over standard SGD, but still struggles with 'Textile Trash' (F1: 0.22) and 'Miscellaneous Trash' (F1: 0.40).

Table 3: Classification Report: SGD with Momentum

Class	Precision	Recall	F1-Score	Support
Cardboard	0.5181	0.6232	0.5658	69
Food Organics	0.7115	0.6066	0.6549	61
Glass	0.7313	0.7778	0.7538	63
Metal	0.7429	0.4370	0.5503	119
Miscellaneous Trash	0.5476	0.3108	0.3966	74
Paper	0.5517	0.8533	0.6702	75
Plastic	0.5404	0.7754	0.6369	138
Textile Trash	0.4118	0.1458	0.2154	48
Vegetation	0.8824	0.9091	0.8955	66
Accuracy			**0.6199**	**713**
Macro Avg	**0.6264**	**0.6043**	**0.5933**	**713**
Weighted Avg	**0.6285**	**0.6199**	**0.6015**	**713**

9.3 Performance Summary

The performance metrics chosen were **Test Accuracy** (the primary indicator of classification performance) and **Test Loss** (an indicator of how well the model's predictions match the true labels).

Table 4: Optimizer Performance Comparison

Optimizer	Test Accuracy	Test Loss
Adam (LR: 1e-4)	**71.95%**	**1.0306**
SGD + Momentum (LR: 0.01)	61.99%	1.8313
Standard SGD (LR: 0.01)	56.38%	2.0864

As shown in Table 4, the **Adam** optimizer significantly outperformed both SGD variants, achieving an accuracy of nearly 72%, compared to 62% (SGDM) and 56

10 Impact of Momentum Parameter

(Corresponds to Q11) The impact of the momentum parameter is clearly demonstrated by comparing the results of standard SGD and SGD with Momentum (SGDM).

- **Standard SGD Accuracy:** 56.38%
- **SGDM (momentum=0.9) Accuracy:** 61.99%

Adding momentum provided a significant performance boost, improving accuracy by over 5.6 percentage points.

Momentum helps the optimizer by accumulating a velocity in directions of persistent, low gradients. This allows it to "roll past" small local minima and navigate "ravines" (areas where the loss landscape is steep in one dimension but flat in another) more effectively. The standard SGD optimizer, lacking this velocity, likely got stuck in a poor local minimum, resulting in significantly worse performance. The loss plots also show that the SGDM validation loss (Figure 3a) is more stable and trends lower than the standard SGD loss (Figure 2a).

11 Model Evaluation

(Corresponds to Q12) This section summarizes the final results of the best-performing model, the **Custom CNN with the Adam optimizer**.

11.1 Test Accuracy

The model achieved the following performance:

- **Test Loss:** 1.0306
- **Test Accuracy:** 0.7195 (71.95%)

11.2 Confusion Matrix

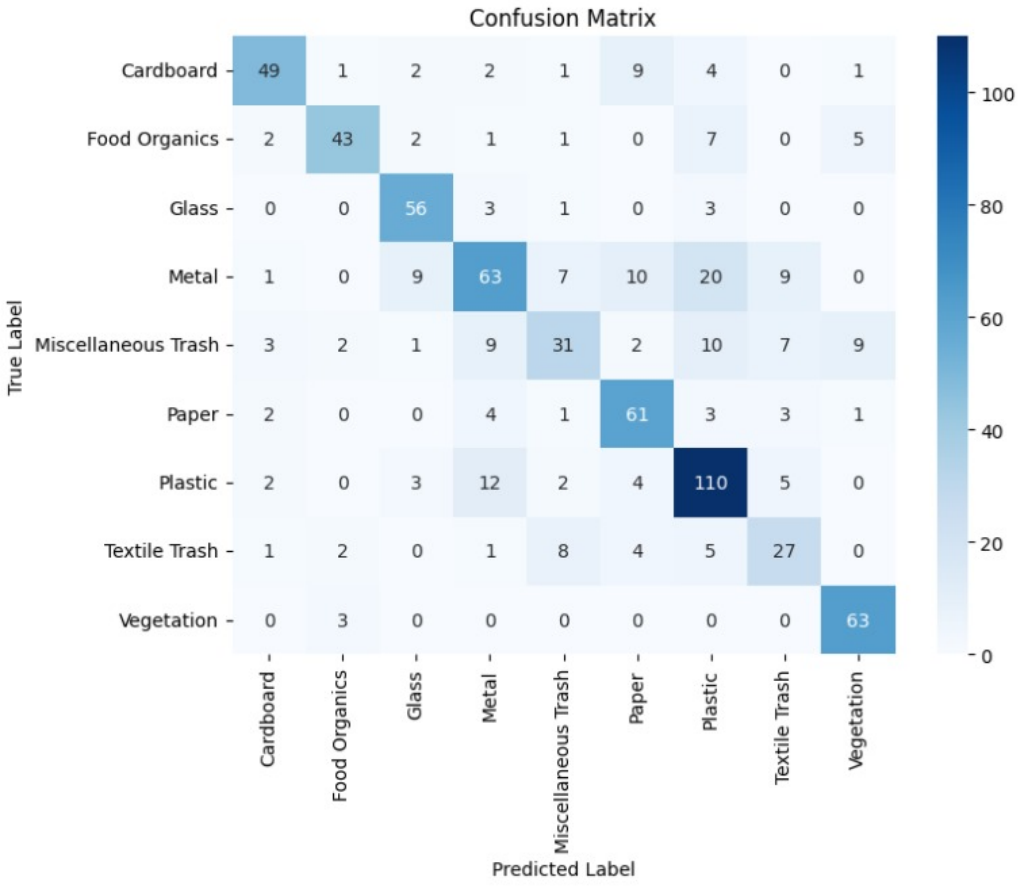


Figure 4: Custom CNN (Adam Opt.): Confusion Matrix on Test Dataset

11.3 Precision, Recall, and F1-Score

Table 5: Custom CNN (Adam Opt.): Classification Report on Test Data

Class	Precision	Recall	F1-Score	Support
Cardboard	0.9020	0.6667	0.7667	69
Food Organics	0.8545	0.7705	0.8103	61
Glass	0.8548	0.8413	0.8480	63
Metal	0.6783	0.6555	0.6667	119
Miscellaneous Trash	0.5932	0.4730	0.5263	74
Paper	0.7000	0.8400	0.7636	75
Plastic	0.6710	0.7536	0.7099	138
Textile Trash	0.4918	0.6250	0.5505	48
Vegetation	0.8769	0.8636	0.8702	66
Accuracy			**0.7195**	**713**
Macro Avg	**0.7358**	**0.7210**	**0.7236**	**713**
Weighted Avg	**0.7285**	**0.7195**	**0.7197**	**713**

11.4 Discussion

The overall test accuracy for the custom CNN using the Adam optimizer is **71.95**%. This result, while solid for a model trained from scratch, reveals specific strengths and weaknesses:

- **Strong Performance (High F1-Score):** The model shows strong performance on classes like **Vegetation** (F1: 0.87), **Glass** (F1: 0.85), and **Food Organics** (F1: 0.81), indicating it can clearly distinguish their features, possibly due to unique colors or shapes.
- **Weak Performance (Low F1-Score):** The model struggles most with **Miscellaneous Trash** (F1: 0.53) and **Textile Trash** (F1: 0.55). This is likely due to the high intra-class variance and ambiguous features (e.g., visual similarity to Paper or Plastic) inherent in these categories, as shown by the confusion matrix in Figure 4.
- **High Precision, Low Recall (Cardboard):** The model is highly confident when classifying an item as Cardboard (Precision: 0.90), but it misses many actual Cardboard items (Recall: 0.67), suggesting that its learned features for this class are highly specific.

12 Comparison with State-of-the-Art Networks

(Corresponds to Q13) For this part, two state-of-the-art pre-trained models were chosen for transfer learning:

- **ResNet50:** A 50-layer Residual Network with "skip connections" that mitigate vanishing gradients and enable very deep architectures
- **DenseNet121:** A 121-layer Densely Connected Convolutional Network where each layer connects to every other layer, promoting feature reuse and improving gradient flow

12.1 Methodology: Fine-Tuning (Feature Extraction)

(Corresponds to Q14) The methodology used is transfer learning through "feature extraction":

1. Models loaded with weights pre-trained on ImageNet dataset
2. All parameters in the convolutional base frozen (`requires_grad = False`)
3. Final classifier layer replaced with new `nn.Linear` layer
4. New layer configured with 9 output units (matching dataset classes)
5. Only the new classifier layer parameters were trained

This approach leverages powerful features learned from 1.2 million ImageNet images while only training a small layer to map these features to our specific classes.

12.2 Environment and Data Loading

The environment switched to **PyTorch** to leverage `torchvision.models`:

- **Data Splitting:** 70% train, 15% validation, 15% test
- **Transforms:** Images resized to 224×224; augmentations (RandomFlip, RandomRotation) applied to training set; ImageNet normalization used
- **DataLoaders:** Batch size of 32
- **Device:** CUDA-enabled GPU

Final Dataset: 3,323 training images, 710 validation images, 719 test images

13 Model 1: ResNet50

13.1 Training (ResNet50)

(Corresponds to Q15, Q16) The ResNet50 classifier was trained for 20 epochs using Adam optimizer (learning rate: 0.001) with Cross-Entropy Loss.

Figure 5: ResNet50: Training and Validation Loss vs. Epochs

Table 6: ResNet50 Training Log (Selected Epochs)

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	1.3630	54.23%	1.0451	62.68%
2	0.8809	70.42%	0.8799	68.73%
5	0.6622	77.19%	0.7116	75.77%
...	...	...	...	...
20	0.4735	83.30%	0.6441	79.30%

13.2 Evaluation (ResNet50)

(Corresponds to Q17) The fine-tuned ResNet50 achieved an **overall accuracy of 81%** on the test set.

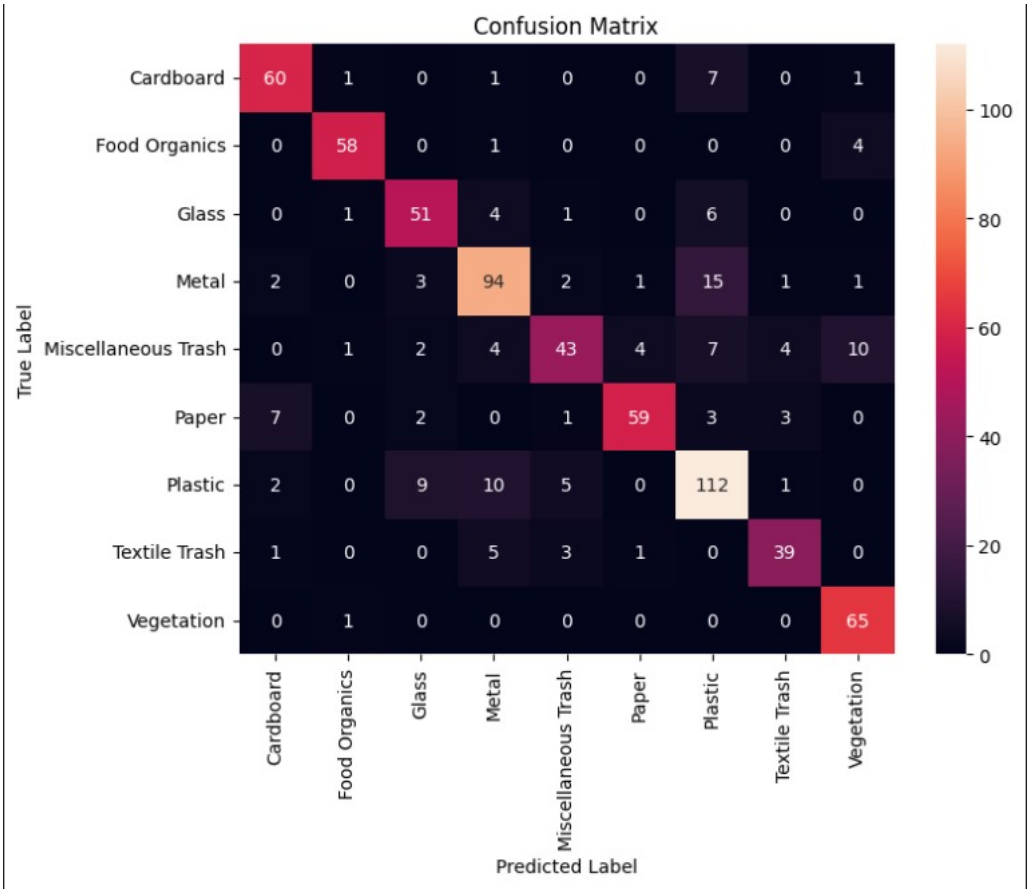


Figure 6: ResNet50: Confusion Matrix on Test Dataset

Table 7: ResNet50: Classification Report on Test Data

Class	Precision	Recall	F1-Score	Support
Cardboard	0.83	0.86	0.85	70
Food Organics	0.94	0.92	0.93	63
Glass	0.76	0.81	0.78	63
Metal	0.79	0.79	0.79	119
Miscellaneous Trash	0.78	0.57	0.66	75
Paper	0.91	0.79	0.84	75
Plastic	0.75	0.81	0.78	139
Textile Trash	0.81	0.80	0.80	49
Vegetation	0.80	0.98	0.88	66
Accuracy			**0.81**	**719**
Macro Avg	**0.82**	**0.81**	**0.81**	**719**
Weighted Avg	**0.81**	**0.81**	**0.81**	**719**

14 Model 2: DenseNet121

14.1 Training (DenseNet121)

(Corresponds to Q15, Q16) DenseNet121 was trained with the same configuration (20 epochs, Adam, LR=0.001), showing even faster convergence.

Table 8: DenseNet121 Training Log (Selected Epochs)

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	1.5529	46.70%	1.1766	62.54%
2	0.9824	69.85%	0.9729	66.90%
5	0.6657	78.48%	0.7475	74.37%
...	...	...	...	...
20	0.4548	84.14%	0.5648	81.27%

14.2 Evaluation (DenseNet121)

(Corresponds to Q17) DenseNet121 also achieved an **overall accuracy of 81%** on the test set.

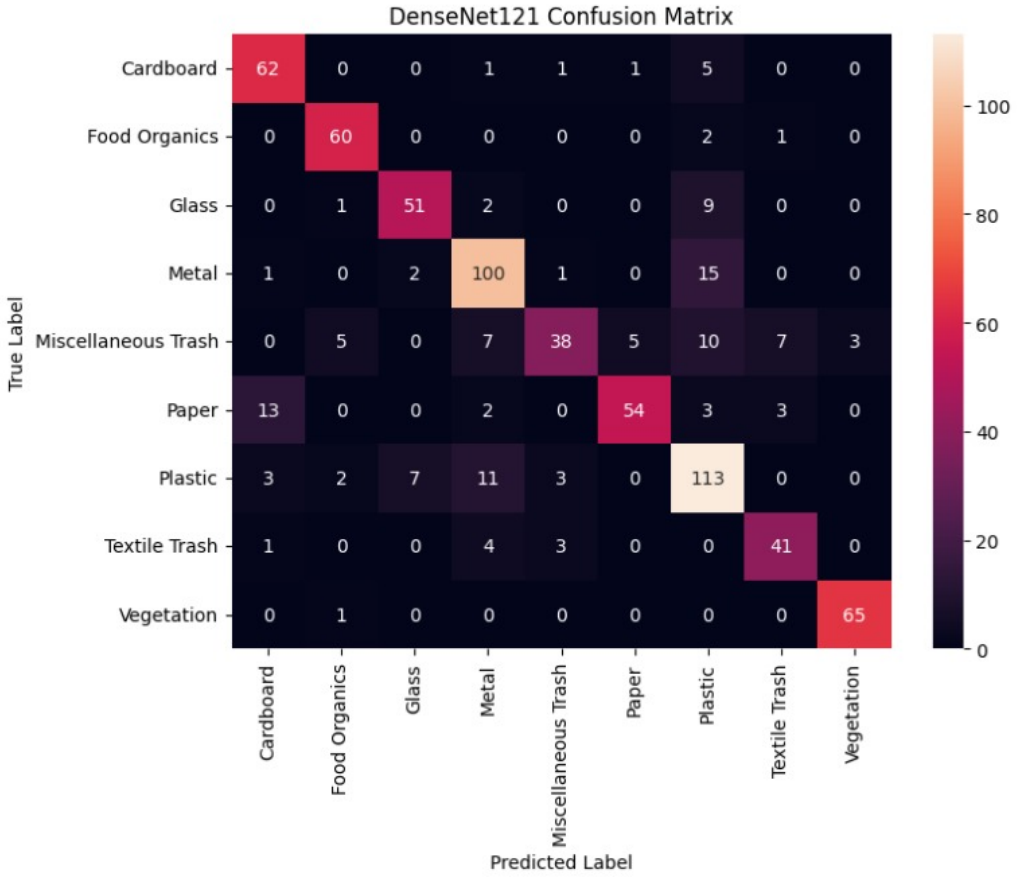


Figure 7: DenseNet121: Confusion Matrix on Test Dataset

Table 9: DenseNet121: Classification Report on Test Data

Class	Precision	Recall	F1-Score	Support
Cardboard	0.78	0.89	0.83	70
Food Organics	0.87	0.95	0.91	63
Glass	0.85	0.81	0.83	63
Metal	0.79	0.84	0.81	119
Miscellaneous Trash	0.83	0.51	0.63	75
Paper	0.90	0.72	0.80	75
Plastic	0.72	0.81	0.76	139
Textile Trash	0.79	0.84	0.81	49
Vegetation	0.96	0.98	0.97	66
Accuracy			**0.81**	**719**
Macro Avg	**0.83**	**0.82**	**0.82**	**719**
Weighted Avg	**0.82**	**0.81**	**0.81**	**719**

15 Final Model Comparison

(Corresponds to Q18)

A direct comparison of test accuracies highlights the fundamental difference between training a model from scratch and leveraging transfer learning. Table 10 presents the final evaluation results for all models tested on the RealWaste dataset.

Table 10: Final Model Performance Comparison on the Test Set

Model	Test Accuracy (%)
Custom CNN (Adam Optimizer)	71.95
ResNet50 (Fine-Tuned)	81.00
DenseNet121 (Fine-Tuned)	81.00

Both pre-trained models, ResNet50 and DenseNet121, achieved identical overall accuracies of **81.00%** on the test set. This represents a notable improvement of **9.05 percentage points** over the best-performing custom CNN model, corresponding to a relative performance gain of approximately 12.6%.

Analysis of the Performance Gap

- **Transfer Learning Advantage:** The superior accuracy of the state-of-the-art (SOTA) models can be attributed to the use of transfer learning. Their convolutional bases, pre-trained on the large-scale ImageNet dataset (over 1.2 million images), have already learned robust, generalized feature representations such as edges, shapes, and textures. The fine-tuning process in this study focused primarily on adapting the final classification layers to the nine RealWaste categories.
- **Training Efficiency:** In contrast, the custom CNN was trained from scratch with random weight initialization on a relatively small dataset. This required the network to learn all low- and high-level features independently, making the process slower and less efficient, ultimately leading to reduced accuracy and generalization capability.

These results clearly demonstrate that for standard image classification tasks where a relevant pre-trained model is available, **transfer learning provides a significant advantage in both performance and efficiency**. It enables faster convergence, requires fewer computational resources, and achieves higher accuracy, making it the preferred approach in practical applications.

16 Trade-offs: Custom vs. Pre-trained Models

(Corresponds to Q19) The decision to use a custom model versus a pre-trained model involves significant trade-offs, summarized below.

16.1 Custom CNN (From Scratch)

- **Advantages:**
  - **Full Control:** The architecture can be designed to be lightweight and fast, which is critical for edge devices or mobile applications.
  - **Domain Specificity:** It can be tailored to unique data types (e.g., 1D sensor data, non-image 2D data) where pre-trained models do not exist.
  - **No Domain Mismatch:** It learns features *only* from the target data, avoiding any potential negative transfer from irrelevant ImageNet features.
- **Limitations:**
  - **Data Hungry:** Requires a very large, diverse, and well-labeled dataset to learn robust features from random initialization.
  - **Computationally Expensive:** Training from scratch is slow and requires significant computational resources and extensive hyperparameter tuning.
  - **Lower Performance:** As seen in this assignment, it is very difficult to match the performance of state-of-the-art models without a massive dataset.

16.2 Pre-trained Model (Transfer Learning)

- **Advantages:**
  - **High Accuracy:** This is the most significant advantage. It provides a massive performance boost by leveraging knowledge from millions of images (**81.00**% vs **71.95**%).
  - **Fast Training:** Training is extremely fast because only the small, final classifier layer is being updated.
  - **Low Data Requirement:** It works very well even with small datasets, as the core feature extraction is already complete.
- **Limitations:**
  - **Inference Speed/Size:** Pre-trained models like ResNet50 or DenseNet121 are very large and computationally "heavy," making them less suitable for resource-constrained environments.

- **Domain Mismatch:** If the target task is very different from ImageNet (e.g., medical X-rays, astronomical data), the pre-trained features may not be optimal and could even hinder performance.
- **Input Constraints:** The model is "stuck" with the original input size (e.g., 224×224) and normalization requirements, which may not be ideal for all tasks.

17 GitHub Profile

The GitHub repository containing all code, commits, and files for this assignment can be found at the following URL:

github.com/john-praveen/EN3150-Assignment-03