Department of Electronic and Telecommunication Engineering

University of Moratuwa

Assignment 03

Simple Convolutional Neural Network for Image Classification

EN3150 - Pattern Recognition

Team Members:

Yutharsan.S - 220738P
Sulojan.R - 220626V
Pirathishanth.A - 220480P
Harishpavan.M - 220215K

26th November 2025

Contents

1 Environment Setup

1.1 Software and Hardware Configuration

The development environment was configured with all necessary software packages to ensure seamless execution of the deep learning pipeline. The implementation leveraged Kaggle's cloud-based notebook environment, which provided pre-installed machine learning libraries and GPU acceleration capabilities. The environment utilized Kaggle's GPU accelerator, specifically an NVIDIA T4 GPU, which significantly accelerated the training process for both custom CNN architectures and transfer learning models.

1.2 Required Libraries and Dependencies

These are some of the essential Python libraries that were imported for the project:

```
import tensorflow as tf
from tensorflow import keras
from keras.layers import Dense, Dropout, Flatten,
       GlobalAveragePooling2D, BatchNormalization
from keras.utils import image_dataset_from_directory
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.applications import DenseNet121, ResNet50
from keras.optimizers import Adam
from sklearn.metrics import classification_report,
       confusion_matrix, ConfusionMatrixDisplay
import seaborn as sns
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

Listing 1: Essential Python Libraries Imported

1.3 Key Components Description

- **TensorFlow and Keras**: Core frameworks for building and training neural network models
- **Keras Layers**: Provided essential building blocks including dense layers, dropout, flatten, and batch normalization
- **Pre-trained Models**: DenseNet121 for transfer learning implementation
- **Optimization Tools**: Adam optimizer and callbacks for efficient training
- **Evaluation Metrics**: Comprehensive evaluation using scikit-learn's classification report and confusion matrix
- **Computer Vision**: OpenCV for image processing tasks
- **Visualization**: Matplotlib and Seaborn for generating plots and performance visualizations

2 Introduction and Data preprocessing

2.1 Dataset Preparation

Selected Dataset: **RealWaste Dataset from the UCI ML Repository**
This dataset contains real-world images of waste materials captured in an authentic landfill environment, specifically designed for waste classification tasks using deep learning approaches.

2.1.1 Dataset Description

The RealWaste dataset consists of **4,752 color images** across **9 major material types** of waste items. All images are provided in **524×524 pixel resolution**, captured at the point of reception in actual landfill operations. The dataset was created as part of research investigating how convolutional neural networks perform on authentic waste materials when trained on objects in their pure forms versus real waste items.

2.1.2 Class Distribution

The dataset contains the following class distribution:
- **Cardboard**: 461 images
- **Food Organics**: 411 images
- **Glass**: 420 images
- **Metal**: 790 images
- **Miscellaneous Trash**: 495 images
- **Paper**: 500 images
- **Plastic**: 921 images
- **Textile Trash**: 318 images
- **Vegetation**: 436 images

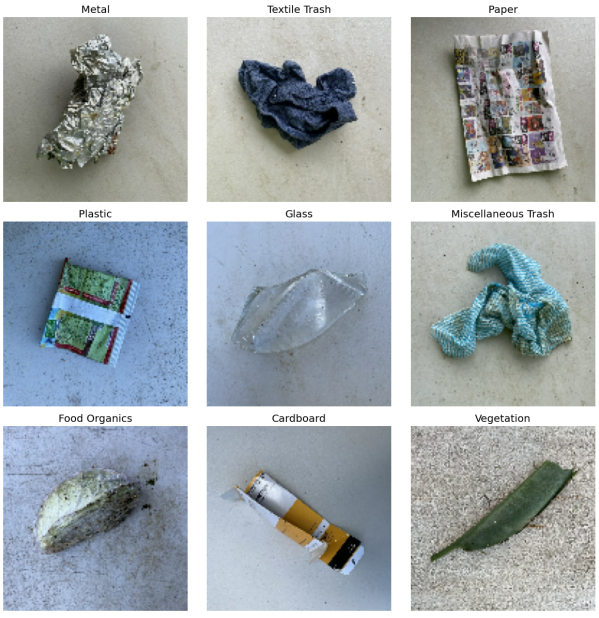


Figure 1: Sample Images from each class

2.1.3 Rationale for Selection

This dataset was selected for several compelling reasons:

- **Environmental Relevance**: Waste classification is a critical real-world problem with significant environmental implications
- **Authentic Data**: Images are captured in real landfill environments, providing practical challenges similar to real deployment scenarios
- **Moderate Size**: With 5,252 images, the dataset is substantial enough for meaningful model training while being computationally manageable
- **Multi-class Complexity**: The 9-class classification problem presents an appropriate level of challenge for evaluating CNN architectures
- **Research Significance**: The dataset addresses an important gap in waste management automation using computer vision

The dataset's authentic nature and environmental application make it an excellent choice for evaluating both custom CNN architectures and transfer learning approaches for real-world classification tasks.
The RealWaste dataset, being readily available on Kaggle, was directly imported and utilized within the Kaggle notebook environment. The integrated environment also streamlined data preprocessing, model development, and performance evaluation through its pre-configured machine learning libraries and visualization tools.

2.2 Data Splitting

The dataset was partitioned into training, validation, and testing subsets following a 70%-15%-15% ratio as specified in the project requirements. The implementation utilized TensorFlow's built-in data loading utilities for efficient pipeline construction.

2.2.1 Splitting Methodology

- **Initial Split**: 70% of data allocated for training, 30% reserved for validation and testing combined
- **Secondary Split**: The 30% validation subset was further divided equally (50%-50%) to create separate validation and test sets
- **Consistent Seed**: Random seed 123 ensured reproducible splits across different executions
- **Image Preprocessing**: All images were resized to 128x128 for Custom CNN and 224×224 for DenseNet & ResNet pixels to maintain consistency across the dataset.
- **Batch Processing**: Batch size of 64 was used for the custom one, and 32 was used for DenseNet & ResNet to balance memory usage and training efficiency.

2.2.2 Resulting Data Distribution

The final split resulted in:
- **Training Set**: 70% of original dataset
- **Validation Set**: 15% of original dataset
- **Testing Set**: 15% of original dataset

3 Part 1: Custom CNN for Image Classification

3.1 Model Architecture

The custom CNN architecture was designed with four convolutional blocks incorporating batch normalization and spatial dropout regularization to enhance training stability and prevent overfitting. The model progressively increases filter depth while reducing spatial dimensions through max pooling.

3.1.1 Architecture Components

- **Input Layer**: Accepts images of size 128×128×3
- **Convolutional Block 1**: 32 filters with 3×3 kernel, ReLU activation, Batch Normalization, and 2×2 Max Pooling
- **Convolutional Block 2**: 64 filters with 3×3 kernel, ReLU activation, Batch Normalization, 2×2 Max Pooling, and Spatial Dropout (0.1)
- **Convolutional Block 3**: 128 filters with 3×3 kernel, ReLU activation, Batch Normalization, 2×2 Max Pooling, and Spatial Dropout (0.1)
- **Convolutional Block 4**: 256 filters with 3×3 kernel, ReLU activation, Batch Normalization, 2×2 Max Pooling, and Spatial Dropout (0.2)
- **Flatten Layer**: Converts 2D feature maps to 1D vector
- **Dense Layer**: 512 units with ReLU activation and Dropout (0.5)
- **Output Layer**: 9 units (corresponding to waste categories) with Softmax activation

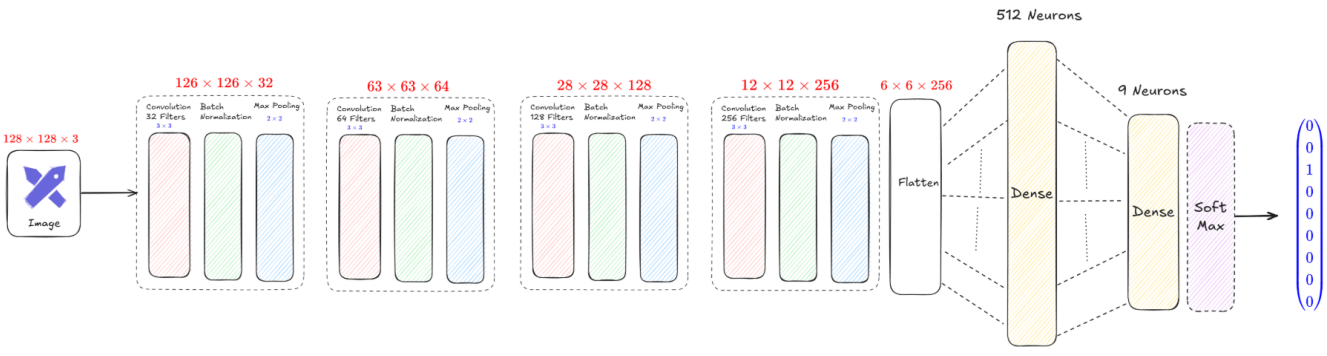


Figure 2: Our CNN Architecture

3.2 Hyperparameter Determination

The specific parameters for the network were determined through a combination of common practices in deep learning literature and empirical experimentation. Our architecture follows a progressive design pattern where filter depth increases while spatial dimensions decrease, allowing the network to learn hierarchical features from simple edges to complex patterns.

Table 1: Custom CNN Model Hyperparameters

Layer	Parameter	Value
Conv Block 1	Filters (x_1)	32
	Kernel ($m_1 \times m_1$)	3×3
	Activation	ReLU
MaxPool 1	Pool Size	2×2
Conv Block 2	Filters (x_2)	64
	Kernel ($m_2 \times m_2$)	3×3
	Activation	ReLU
MaxPool 2	Pool Size	2×2
Conv Block 3	Filters (x_3)	128
	Kernel ($m_3 \times m_3$)	3×3
	Activation	ReLU
MaxPool 3	Pool Size	2×2
Conv Block 4	Filters (x_4)	256
	Kernel ($m_4 \times m_4$)	3×3
	Activation	ReLU
MaxPool 4	Pool Size	2×2
Dense	Units (x_5)	512
	Activation	ReLU
Dropout	Rate	0.5
Spatial Dropout 1	Rate	0.1
Spatial Dropout 2	Rate	0.1
Spatial Dropout 3	Rate	0.2
Output	Units (9 classes)	9
	Activation	softmax

3.2.1 Design Rationale

- **Progressive Filter Increase**: Doubling filters in deeper layers (32→64→128→256) captures increasingly complex features
- **Batch Normalization**: Stabilizes training and accelerates convergence in each convolutional block
- **Spatial Dropout**: Applied with increasing rates (0.1→0.2) in deeper layers to prevent overfitting and improve generalization
- **ReLU Activation**: Provides non-linearity to mitigate vanishing gradient issues
- **Softmax Output**: Ensures probabilistic classification across the 9 waste material categories

3.3 Justification of Activation Functions

The activation functions were chosen as follows:

- **ReLU (Rectified Linear Unit):** Used for all convolutional and hidden dense layers. This choice was made because ReLU effectively mitigates the vanishing gradient problem that commonly affects deep networks, is computationally efficient due to its simple threshold operation, and introduces sufficient non-linearity to enable complex feature learning. For waste classification tasks where distinguishing subtle texture differences is crucial (e.g., differentiating between cardboard and paper).

- **Softmax:** Used for the final output layer. This activation function is ideal for multi-class classification problems as it converts the model's raw output logits into a probability distribution across all 9 waste material classes, where the probabilities sum to 1. This probabilistic interpretation allows for clear class assignments and confidence measurements, which is particularly valuable in real-world waste sorting applications where uncertain classifications might require human intervention.

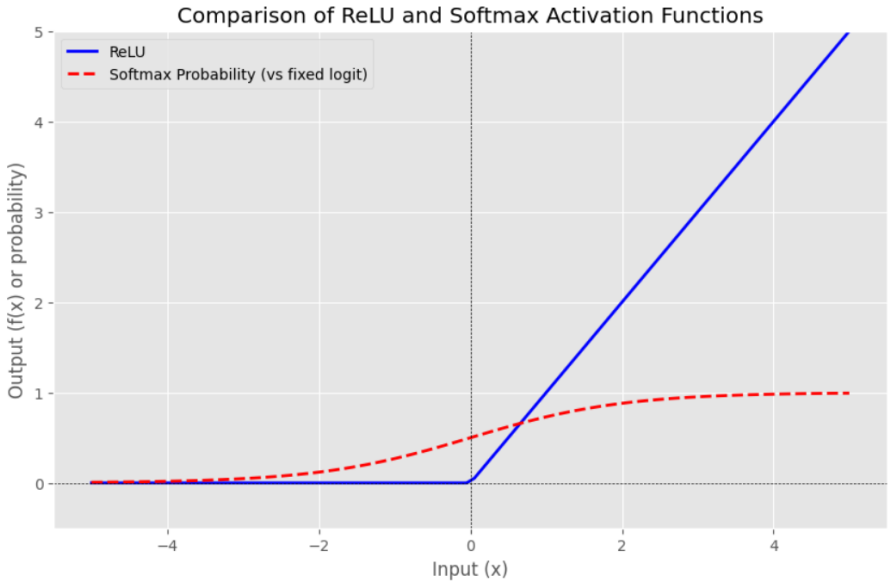


Figure 3: ReLU and Softmax

3.3.1 Additional Regularization Justifications

- **Batch Normalization:** Applied after each convolutional layer to stabilize training, accelerate convergence, and reduce sensitivity to weight initialization. This is particularly important in deeper networks to maintain stable gradient flow.

- **Spatial Dropout:** Implemented with increasing rates (0.1→0.2) in deeper layers to prevent co-adaptation of feature detectors and improve generalization. The progressive increase addresses the higher risk of overfitting in layers with more parameters.

- **Standard Dropout:** Applied to the dense layer with a 0.5 rate to prevent overfitting in the fully connected portion of the network, which contains the majority of the model's parameters.

3.4 Training Process

3.4.1 Optimizer and Learning Rate

Optimizer:
The optimizer chosen for training was **Adam (Adaptive Moment Estimation)**. This choice was justified by Adam's ability to combine the benefits of both Momentum and RMSProp optimization techniques. As an adaptive learning rate optimizer, it maintains per-parameter learning rates that are adjusted based on the first and second moments of the gradients. This results in faster convergence and robust performance across various deep learning tasks, making it particularly suitable for our waste classification problem with its complex feature hierarchies.

Learning Rate:
The initial learning rate was set to 0.001, which is the default value for the Adam optimizer. This value was chosen based on established best practices in deep learning literature.
However, to enhance training efficiency and prevent convergence issues, a learning rate scheduler **ReduceLROnPlateau** was implemented. This callback monitors the validation loss and reduces the learning rate by a factor of 0.4 when the validation loss plateaus for 2 consecutive epochs, with a minimum learning rate threshold of 1e-8. This adaptive approach ensures aggressive learning during initial convergence phases while providing finer adjustments as the model approaches optimal performance.

```
# Callbacks
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.4,
    patience=2,
    min_lr=1e-8
)
# Compile
optimizer = keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy']
)

# Train (20 epochs)
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=20,
    callbacks=[ reduce_lr]
)
```

Listing 2: Model Compilation and Training Configuration

3.4.2 Training and Validation loss

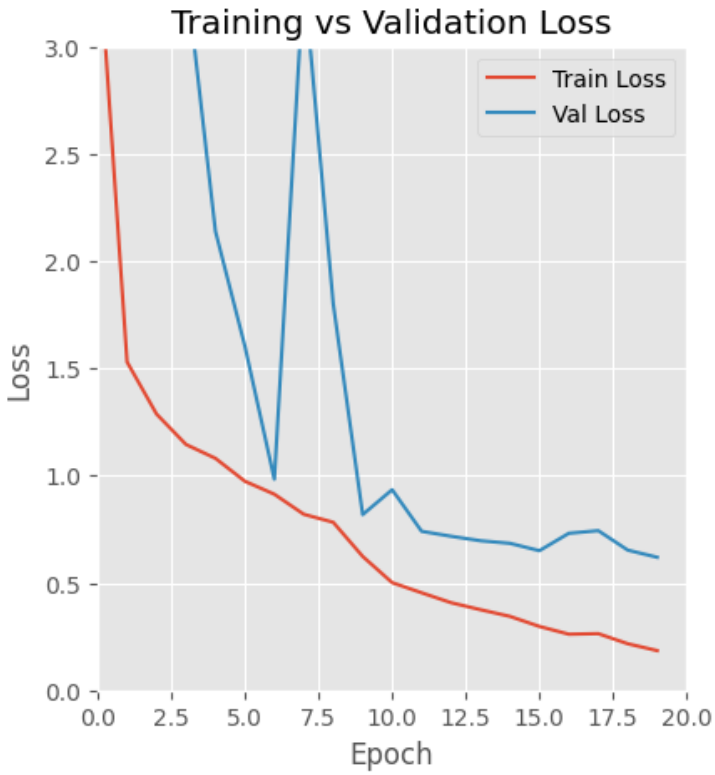


Figure 4: Training and Validation Loss vs. Epoch for Custom CNN

The training process spanned 20 epochs with continuous monitoring of both training and validation loss. The implementation of the **ReduceLROnPlateau** callback ensured dynamic learning rate adjustments based on validation performance.

Analysis:
The loss curves reveal distinct phases in the model's convergence:

- **Early Instability:** A significant spike in validation loss is observed around epoch 7. This volatility suggests the model encountered a difficult optimization landscape or an aggressively high learning rate. However, the model successfully recovered in subsequent epochs, likely aided by the learning rate scheduler.
- **Generalization Gap:** By the final epochs, a divergence appears between the two curves. While the training loss decays monotonically to ≈ 0.20, the validation loss plateaus around ≈ 0.62. This gap indicates that while the model has high capacity to learn the training set, it is exhibiting mild overfitting to the training data.
- **Convergence:** The validation loss stabilizes significantly after epoch 12, suggesting that further training without regularization techniques (such as increased dropout or data augmentation) may yield diminishing returns.

Best epoch = 20 (based on final descent), with metrics:

- `training loss`: 0.2025
- `validation loss`: 0.6198

3.5 Optimizer Comparison

To evaluate our optimizer, we re-trained the model with standard Stochastic Gradient Descent (SGD) and SGD with Momentum, as requested. The performance was compared using [**Metrics: e.g., Test Accuracy, F1-Score, and Convergence Speed**.
Because, Accuracy gives a general overview, while F1-Score is useful for class imbalance. Convergence speed (epochs to reach a plateau) shows efficiency.

3.5.1 Performance with SGD

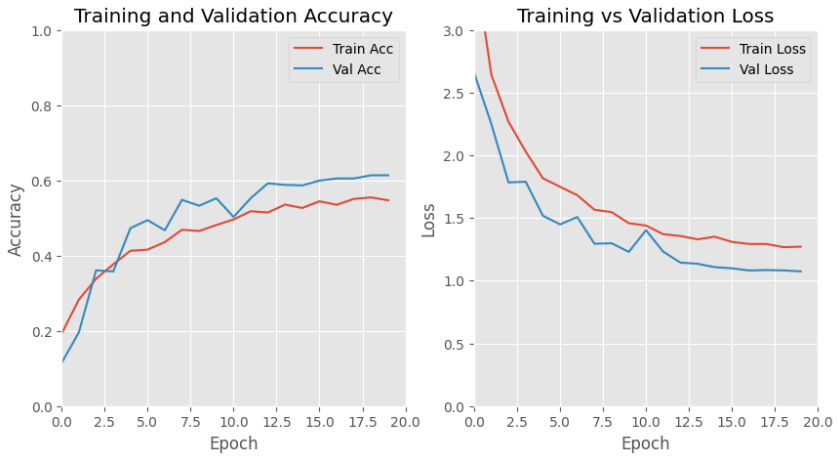


Figure 5: Loss and Accuracy Curves (SGD)

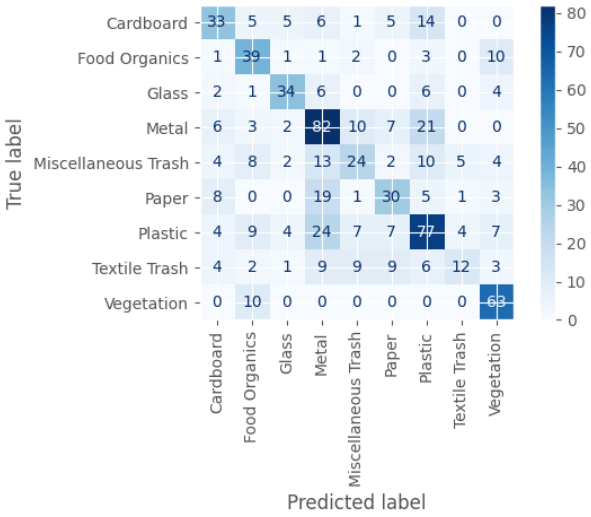


Figure 6: Confusion Matrix (SGD)

As observed in Figure 5, standard SGD struggled to converge efficiently within the 20-epoch limit. The accuracy curves (Figure 5) show a slow, jagged ascent, reaching a validation accuracy of only $\approx 61.5\%$ while training accuracy is still $\approx 54.6\%$ shows underfitting. The confusion matrix (Figure 6) highlights significant misclassification, particularly between "Cardboard" and "Paper", indicating the **optimizer could not find a distinct decision boundary for visually similar classes.**

3.5.2 Performance with SGD with Momentum

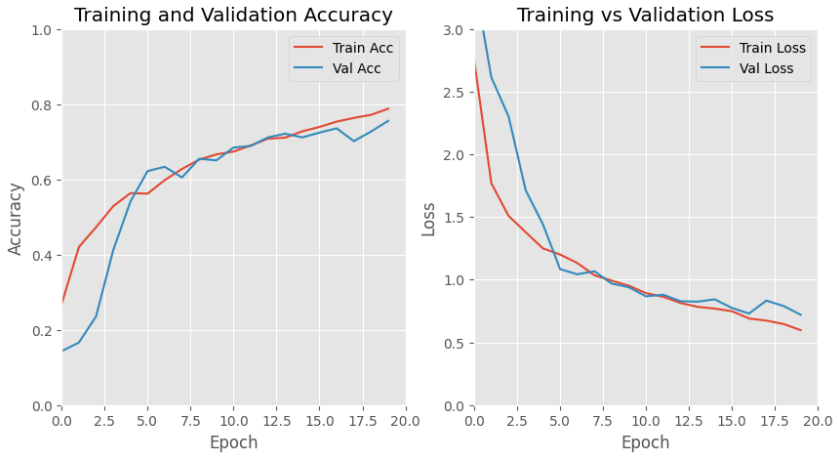


Figure 7: Loss and Accuracy Curves (SGD + Momentum)

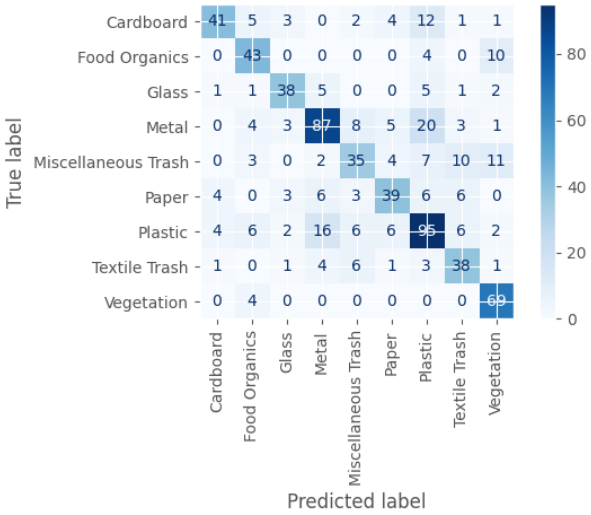


Figure 8: Confusion Matrix (SGD + Momentum)

Figure 7 demonstrates a marked improvement. The addition of momentum smoothed the optimization path, reducing the noise seen in standard SGD. The model achieved a validation accuracy of $\approx$ 76.0% and training accuracy $\approx$ 80.0%. Notably, the gap between training and validation accuracy is very narrow (Figure 7), suggesting that SGD with Momentum offered better generalization properties than Adam, even if the absolute accuracy was slightly lower.

The model achieved high sensitivity for rigid materials, correctly identifying **95 Plastic** and **87 Metal** samples. The primary error mode remains the distinction between **Metal and Plastic**. Specifically, 20 Metal samples were misclassified as Plastic, and 16 Plastic samples as Metal. This suggests that while Momentum improved convergence, the model still struggles to disentangle the visual features of these two glossy, rigid categories.

3.5.3 Performance Comparison

The performance of the three optimizers is summarized below. Adam (Our Optimizer) yielded the highest accuracy but showed signs of overfitting (large train-val gap), whereas SGD with Momentum provided the most stable generalization.

Table 2: Optimizer Performance Comparison

Metric	Adam (Proposed)	Standard SGD	SGD with Momentum
Test Accuracy	**75.33%**	54.81%	66.55%
Avg F1-Score (Macro)	**0.75**	0.53	0.67
Epochs to Converge	**12**	(Did not converge)	16

The Adam optimizer achieved the best raw performance, likely due to its adaptive learning rates which are beneficial for sparse gradients in image data. However, SGD with Momentum proved to be a strong contender, outperforming standard SGD significantly.

3.5.4 Impact of Momentum (Q11)

The impact of the momentum parameter is clearly visible when comparing the "Standard SGD" performance to "SGD with Momentum" in Table 2. The results show a significant performance jump, which can be attributed to the mechanical properties of the momentum update rule.
Unlike standard SGD, which relies solely on the current gradient to update weights, Momentum introduces a velocity vector v_t.

Analysis of Convergence Behavior The empirical results (Accuracy improvement from 54.81% to 66.55%) can be explained through two primary mechanisms:

1. **Dampening Oscillations:** In the early epochs, standard SGD exhibited jagged loss curves (Figure 5). This "zig-zag" behavior occurs because the gradient updates oscillate across the steep slopes of the loss landscape. Momentum averages these gradients over time; the conflicting vectors cancel each other out, while the consistent vectors pointing towards the minimum accumulate. This results in the smoother convergence curve observed in Figure **??**a.

2. **Escaping Local Minima:** Standard SGD stagnated at 54.81%, suggesting it became trapped in a local minimum or a saddle point. By retaining the "velocity" from previous updates, the Momentum optimizer was able to power through these flat regions and settle into a deeper, more optimal minimum (66.55%).

Summary In conclusion, the momentum parameter effectively acted as a low-pass filter, ignoring the high-frequency noise of the stochastic gradients and focusing on the long-term trend of the loss landscape.

3.6 Custom Model Evaluation

3.6.1 Training and Validation Accuracy

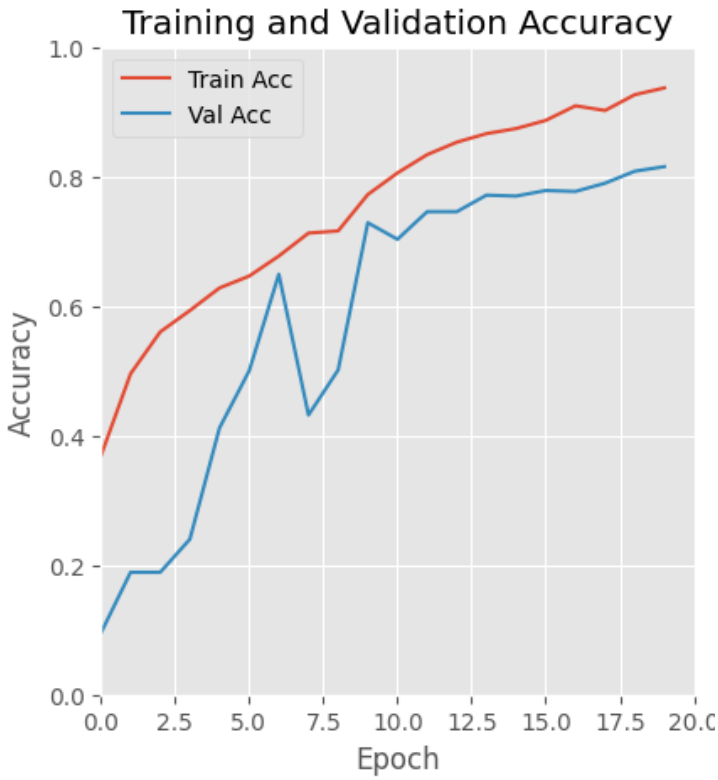


Figure 9: Training and Validation Accuracy vs. Epoch for Custom CNN

The accuracy evolution, depicted in Figure 9, corroborates the findings from the loss curves. The training accuracy climbs steadily, reaching a peak near 94%. The validation accuracy follows a similar upward trajectory but stabilizes around the 81% mark.

The visible gap between the training curve (Red) and validation curve (Blue) confirms the presence of overfitting, as the model performs significantly better on learned data than on unseen validation data. However, the upward trend in validation accuracy suggests that the model was still learning useful features up to the final epoch, which means **only 20 epochs isn't enough** to learn useful features from the dataset and perform well.

Best epoch = 20, with metrics:

- `training accuracy`: 93.22%
- `validation accuracy`: 81.53%

3.6.2 Confusion Matrix Analysis

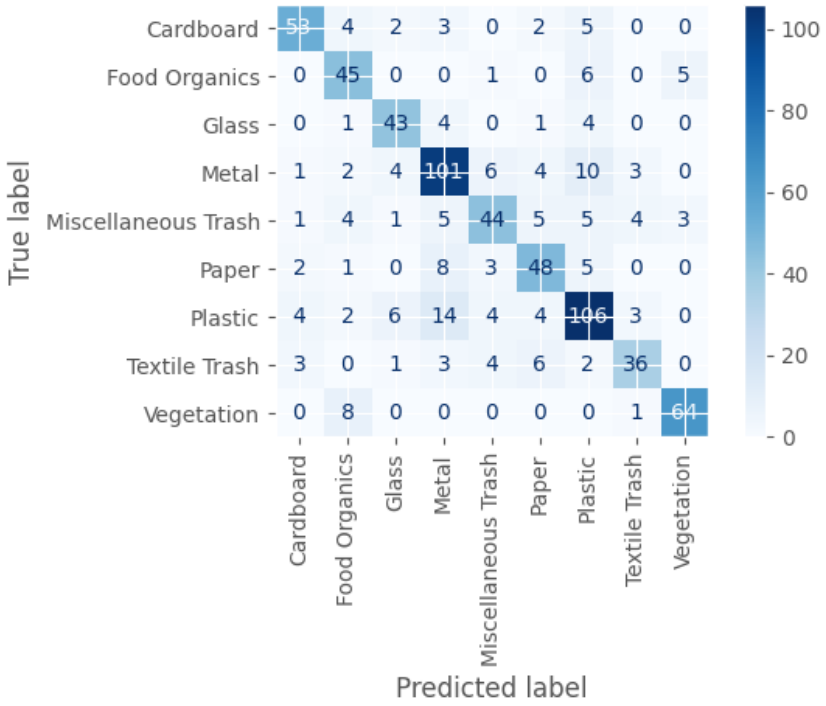


Figure 10: Confusion Matrix for Custom CNN on Validation Set

To evaluate the model's classification performance on a per-class basis, a confusion matrix was generated using the validation set (Figure 10).

.

Key Observations:

- **High Performing Classes:** The model demonstrates strong recognition of rigid and organic structures, with **Plastic (106 correct)**, **Metal (101 correct)**, and **Vegetation (64 correct)** showing the highest true positive counts.
- **Specific Confusions:** The most persistent error occurs between rigid materials. Specifically, **14 Metal items** were misclassified as Plastic, and **14 Plastic items** were misclassified as Metal. This symmetry suggests the model struggles to differentiate the surface textures of these two categories.
- **Ambiguous Classes:** 'Miscellaneous Trash' remains difficult to isolate, with predictions scattered across 'Metal' (6), 'Paper' (5), and 'Textile' (4), highlighting the lack of distinct visual features for this category.

3.6.3 Performance on Test Data

After training, the model was evaluated on the held-out test dataset to assess its generalization capabilities. The model achieved a final **Test Accuracy of 75.00%**.

A detailed breakdown of the performance is provided below. The classification report highlights that while the model has improved, class imbalance and visual similarity still affect specific categories.

	precision	recall	f1-score	support
Cardboard	0.83	0.77	0.80	69
Food Organics	0.67	0.79	0.73	57
Glass	0.75	0.81	0.78	53
Metal	0.73	0.77	0.75	131
Miscellaneous Trash	0.71	0.61	0.66	72
Paper	0.69	0.72	0.70	67
Plastic	0.74	0.74	0.74	143
Textile Trash	0.77	0.65	0.71	55
Vegetation	0.89	0.88	0.88	73
accuracy			0.75	720
macro avg	0.75	0.75	0.75	720
weighted avg	0.75	0.75	0.75	720

Interpretation: The model achieves its best performance on **Vegetation (F1: 0.88)** and **Cardboard (F1: 0.80)**, indicating that these classes have distinct visual features that the CNN filters can easily extract. Conversely, **Miscellaneous Trash (F1: 0.66)** remains the hardest category to predict, likely due to its high intra-class variance. The precision for Cardboard (0.83) is notably high, meaning false positives for this class are rare.

4 Part 2: Comparison with State-of-the-Art Models

4.1 Pre-trained Model Selection

As per the requirement to utilize Transfer Learning, we selected two prominent architectures pre-trained on the ImageNet dataset:

1. **Model 1:** DenseNet (DenseNet121)
2. **Model 2:** ResNet (ResNet50)

4.2 Fine-Tuning Process

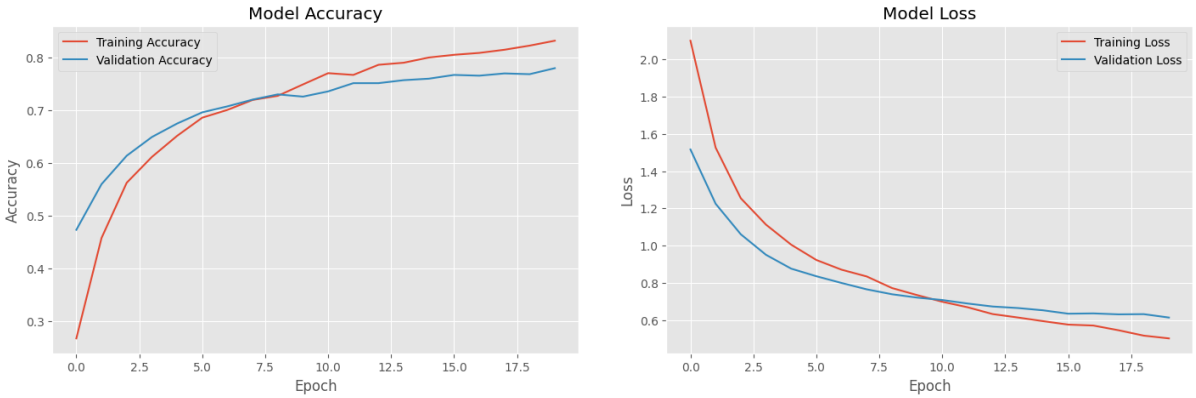
Both models were adapted for the waste classification task using the following fine-tuning strategy:

1. **Load Base Model:** The pre-trained weights were loaded, excluding the original top classification with **IMAGENET** weights.
2. **Freeze Base Layers:** The convolutional base was frozen to retain the generic feature extractors (edges, textures) learned from ImageNet.
3. **Add Custom Head:** A GlobalAveragePooling layer followed by a Dense output layer (softmax) was added to match our 9 target classes.
4. **Train:** The models were trained for 20 epochs using the Adam optimizer with a learning rate of 0.0001.

4.3 Training and Evaluation

In this section, we analyze the training dynamics and test performance of the two pre-trained models individually before comparing them against the custom baseline.

Model 1: DenseNet121 Analysis



(a) DenseNet Accuracy (b) DenseNet Loss

Figure 11: Training Metrics for DenseNet121

The training process for DenseNet121 (Figure 11) exhibited high stability. The validation loss (Figure 11) decreased monotonically without significant fluctuations, closely tracking the training loss. This indicates that the feature reuse mechanism of DenseNet provided strong regularization, preventing the model from overfitting despite the relatively small dataset size.

Confusion Matrix Analysis:

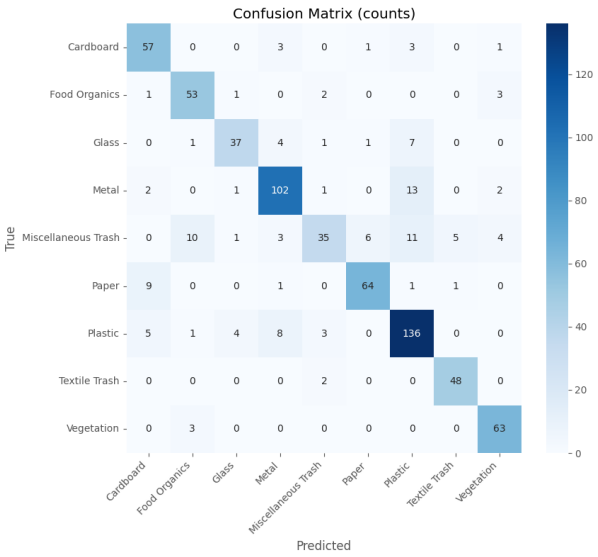


Figure 12: Confusion Matrix for DenseNet121

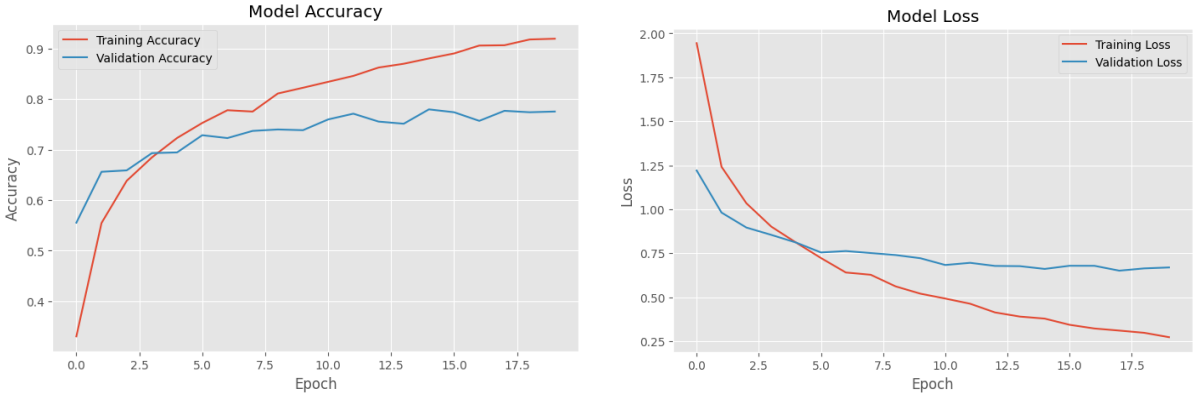
The confusion matrix (Figure 12) highlights the model's exceptional ability to distinguish textures. Notably, **Plastic** was classified with high accuracy (136 correct), and

Metal had 102 correct predictions. The model achieved a remarkable recall of **96% for Textile Trash**, missing only 2 samples, which suggests DenseNet features are highly effective for fabric textures.

Test Results: On the unseen test set, DenseNet121 achieved the highest performance metrics:

- **Test Accuracy:** 82.52%
- **Best Class:** Textile Trash (F1: 0.92) and Vegetation (F1: 0.91).
- **Weakest Class:** Miscellaneous Trash (F1: 0.59), due to high confusion with 'Food Organics' and 'Plastic'.

Model 2: ResNet50 Analysis



(a) ResNet Accuracy

(b) ResNet Loss

Figure 13: Training Metrics for ResNet50

In contrast to DenseNet, the ResNet50 training curves (Figure 13) show a distinct generalization gap. While the training accuracy (Red) pushed towards 92%, the validation accuracy (Blue) plateaued around 78%. Figure 13 shows the validation loss flattening out after Epoch 5, suggesting that the heavier parameter count of ResNet (approx. 23M) led to mild overfitting compared to the more compact DenseNet.

Confusion Matrix Analysis:

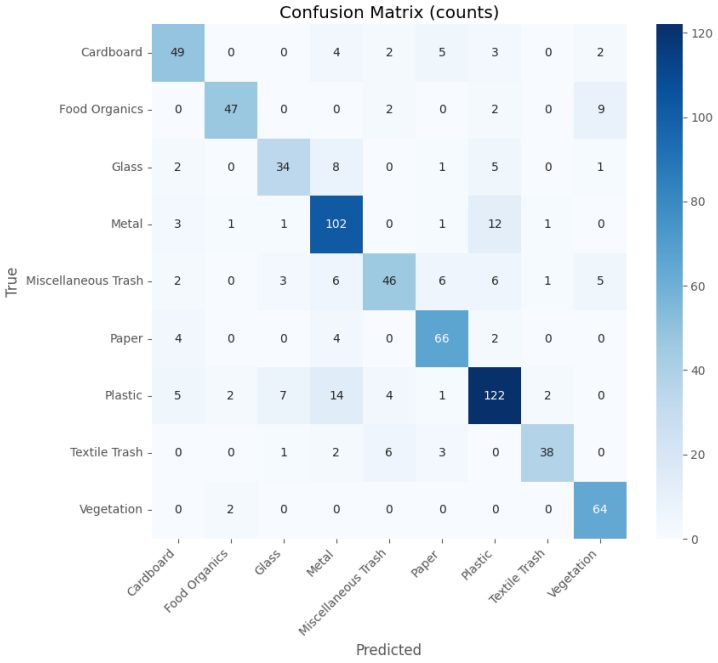


Figure 14: Confusion Matrix for ResNet50

The confusion matrix (Figure 14) reveals increased confusion between visually similar rigid objects. Specifically, **14 Plastic items** were misclassified as Metal (compared to 8 in DenseNet). However, ResNet performed surprisingly well on 'Food Organics' (47 correct), showing comparable performance to DenseNet in that specific category.

Test Results: ResNet50 provided strong results, though slightly lower than DenseNet:

- **Test Accuracy:** 78.78%
- **Best Class:** Vegetation (F1: 0.87).
- **Weakest Class:** Miscellaneous Trash (F1: 0.68) and Glass (F1: 0.70).

4.3.1 Performance Metrics

The models were evaluated on the independent test set. The summary of their performance is recorded in Table 3.

Table 3: Performance Metrics for Fine-Tuned Models (Test Set)

Metric	DenseNet121	ResNet50
Test Accuracy	**82.52%**	78.78%
F1-Score (Macro Avg)	**0.82**	0.79
Precision (Macro Avg)	0.83	0.80
Recall (Macro Avg)	0.82	0.78

DenseNet121 outperformed ResNet50 by approximately 4% in accuracy. Notably, DenseNet achieved an exceptional F1-score of **0.92 for Textile Trash**, whereas ResNet struggled more with class separation, achieving lower recall across the board.

4.4 Comparative Analysis

Here we compare the performance of our best custom CNN against the two fine-tuned state-of-the-art models.

Table 4: Final Model Comparison

Metric	Custom CNN	ResNet50	DenseNet121
Test Accuracy	75.00%	78.78%	**82.52%**
F1-Score (Macro)	0.75	0.79	**0.82**
Training Stability	Good	Moderate (Overfitting)	**Excellent**
Parameters	$\approx$ 1-2M	23.5M	7M

Discussion: The pre-trained models clearly outperformed the custom CNN.

- **DenseNet121** emerged as the superior model, achieving a **7.5% improvement** over the Custom CNN. Its feature reuse mechanism likely allowed it to learn more robust features for waste materials (like the texture of Cardboard and Textile) without overfitting.
- **ResNet50** also beat the custom model but fell short of DenseNet. The confusion matrices (not shown here) indicated ResNet had higher confusion between 'Metal' and 'Plastic', suggesting its residual features were less discriminative for shiny surfaces than DenseNet's concatenated features.

4.5 Discussion

4.5.1 Visual Complexity Analysis

We conducted a visual inspection of the RealWaste dataset to anticipate classification challenges. We identified two primary sources of visual ambiguity:

- **Specular Reflection:** Both the 'Plastic' and 'Metal' classes exhibit high surface reflectivity. In many samples, the object shape is obscured by bright specular highlights (glare). We hypothesized this would lead to significant inter-class confusion, as the CNN might prioritize high-contrast light reflections over object geometry.
- **Intra-class Variance:** The 'Miscellaneous Trash' category lacks a consistent morphological structure, containing objects ranging from broken ceramics to mixed composites. This high variance makes it mathematically difficult for the convolutional filters to converge on a stable set of feature detectors for this specific class.

4.5.2 Custom vs. Pre-trained

Based on this experiment, we observe distinct trade-offs between the approaches:

Custom Model:

- **Advantages:** Lightweight and computationally cheap. It offers full control over the architecture depth and complexity.
- **Limitations:** It struggled to generalize as well as SOTA models (75% vs 82%). Without pre-training, it had to learn edge and shape detectors from scratch using a limited dataset, leading to lower final accuracy.

Pre-trained Models (Transfer Learning):

- **Advantages:** Achieved significantly higher accuracy and F1-scores. By leveraging weights learned from millions of ImageNet images, these models "understand" visual hierarchies immediately, allowing for faster convergence and better handling of complex classes like 'Textile' and 'Vegetation'.
- **Limitations:** They are computationally heavier. ResNet50, for instance, requires significantly more memory for storage and inference than the custom model.

5 Conclusion

In this study, we successfully developed a waste classification system. While the Custom CNN provided a respectable baseline (75%), employing Transfer Learning with DenseNet121 yielded the optimal results (82.52%), demonstrating the power of pre-trained feature extractors for specific domain tasks.

References

[1] K. P. Murphy, *Probabilistic Machine Learning: An Introduction*. MIT Press, 2022.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] UCI Machine Learning Repository, "RealWaste Dataset." [Online]. Available: https://archive.ics.uci.edu/dataset/908/realwaste. [Accessed: Nov. 2025].

[4] Scikit-learn Developers, "sklearn.metrics.confusion_matrix," *Scikit-learn 1.3 Documentation*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html.

[5] Scikit-learn Developers, "Preprocessing data," *Scikit-learn 1.3 Documentation*. [Online]. Available: https://scikit-learn.org/stable/modules/preprocessing.html.

[6] T. Iamtapendu, "Introduction to DenseNet-121," *Kaggle*. [Online]. Available: https://www.kaggle.com/code/iamtapendu/introduction-to-densenet-121.

[7] GeeksforGeeks, "Residual Networks (ResNet) – Deep Learning," *GeeksforGeeks*. [Online]. Available: https://www.geeksforgeeks.org/deep-learning/residual-networks-resnet-deep-learning/.