Assignment 03 Report

EN3150 – Pattern Recognition

Simple Convolutional Neural Network for Image Classification

`GitHub Repository (With multiple branches)`

Group Members:
Gunathilaka K.L. — 220197E
Jayasekara S.P.R — 220257N
Sadaruwan J. — 220498D
Wickramasinghe S.D. — 220701X

Department of Electronic and Telecommunication Engineering
University of Moratuwa

November 26, 2025

Contents

1 Introduction

1.1 Q1: Problem Statement and Environment Setup

Question: Set up the software environment and briefly state the problem you are solving.

Answer:

The goal of this assignment is to build and analyse a convolutional neural network (CNN) for image classification. We focus on automatic waste sorting, where the model has to classify real-world waste images into several categories. An accurate waste classifier can support recycling plants and smart waste-management systems by reducing manual sorting effort and improving recycling efficiency.

All experiments were carried out in Python using the PyTorch deep learning framework. The main libraries used were:

- **PyTorch / Torchvision** – model definition, training loops, data augmentation, pretrained models.
- **NumPy / Pandas** – numerical operations and dataset statistics.
- **Matplotlib / Seaborn** – plotting training curves, confusion matrices and comparison figures.
- **Scikit-learn** – computing accuracy, precision, recall, F1 score and confusion matrices.

Training was carried out on a GPU-enabled environment (Google Colab / CUDA) when available, with automatic fallback to CPU. Random seeds were fixed for reproducibility, and all scripts were tracked in the linked GitHub repository.

1.2 Q2 & Q3: Dataset Preparation and Splits

Question: Prepare a suitable image dataset, describe it, and split it into training, validation and testing sets using a 70/15/15 split.

Answer:

For this assignment we used the **RealWaste** image dataset, which was collected in an actual landfill environment. The dataset contains 4,752 RGB images of size 524×524 pixels, divided into nine waste categories:

- 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)

There is a moderate class imbalance: *Plastic* is the most frequent class, while *Textile Trash* is the least represented. To follow the 70/15/15 requirement, the dataset was split as follows:

- **Training set (70%):** 3,323 images
- **Validation set (15%):** 710 images
- **Test set (15%):** 719 images

The splits were stratified, so that each subset preserves approximately the same class distribution as the original dataset. The lists of file paths for each split were saved to disk for reproducible experiments.

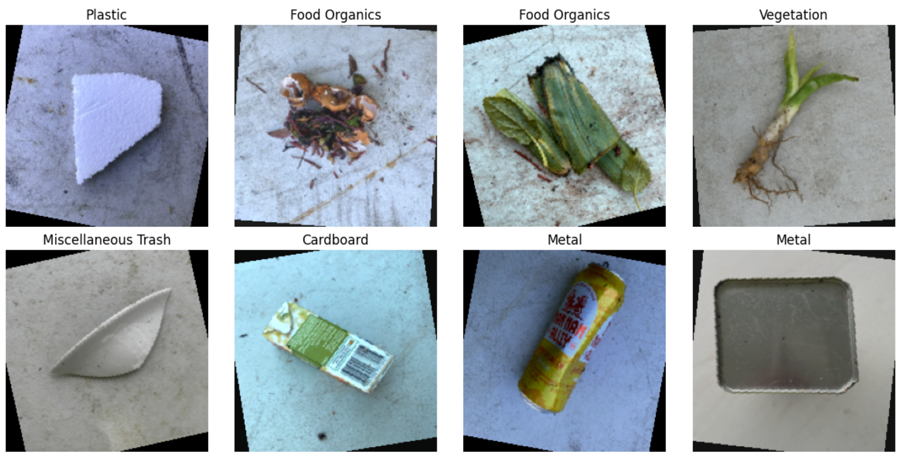

Figure 1: Example images from the RealWaste dataset covering different material classes.

Before feeding images to the network, all images were resized to 224×224 pixels, converted to tensors and normalized using the standard ImageNet mean and standard deviation.

2 Part 1: Custom CNN Implementation

2.1 Q4: CNN Architecture Design

Question: Build a CNN model using convolutional and max-pooling layers, followed by fully connected layers and dropout, without using normalization layers.

Answer:

Following the design instructions, we implemented a relatively shallow CNN composed of three convolution + max-pooling blocks followed by a small fully connected classifier. No batch normalization layers were used, because the network depth is moderate and the benefits of normalization are less pronounced in such shallow architectures.

Architecture Overview

The network processes 224×224 RGB images and has the following structure:

Table 1: Custom CNN architecture for RealWaste classification

Layer	Details	Output Size
Input	$3 \times 224 \times 224$ RGB image	$3 \times 224 \times 224$
Conv Block 1	Conv(3→32, 3×3, padding 1) + ReLU + MaxPool(2×2)	$32 \times 112 \times 112$
Conv Block 2	Conv(32→64, 3×3, padding 1) + ReLU + MaxPool(2×2)	$64 \times 56 \times 56$
Conv Block 3	Conv(64→128, 3×3, padding 1) + ReLU + MaxPool(2×2)	$128 \times 28 \times 28$
Flatten	–	100,352 features
FC1	Linear(100,352→256) + ReLU	256
Dropout	$p = 0.4$	256
Output Layer	Linear(256→9) (logits for 9 classes)	9

Overall, the network has a few million trainable parameters, with most of them concentrated in the first fully connected layer due to the large flattened feature vector. This is still smaller than typical ImageNet-scale models and is therefore suitable for experimentation on a single GPU.

2.2 Q5: Choice of Network Hyperparameters

Question: Specify and justify the kernel sizes, number of filters, fully connected layer size and dropout rate.

Answer:

Convolutional kernel sizes. All convolutional layers use 3×3 kernels with padding of 1. Stacking small kernels is a common design choice because it increases the effective receptive field while keeping the number of parameters manageable (similar to the VGG family). 3×3 kernels are expressive enough to capture local edges and textures in waste images.

Number of filters. The number of filters increases with depth: $32 \to 64 \to 128$. Early layers capture general low-level patterns (edges, colour blobs), while deeper layers learn higher-level shapes and textures that are specific to materials such as glass, plastic or cardboard. Doubling the channels at each stage is a simple way to gradually increase representational capacity without exploding the parameter count.

Pooling strategy. Each block is followed by a 2×2 max-pooling layer that halves the spatial resolution. Pooling layers reduce the computational cost and provide a degree of translation invariance, which is important because the position of the waste object within the image is not fixed.

Fully connected layer. The flattened feature map is passed to a dense layer with 256 units. This size was chosen as a compromise between expressiveness and overfitting risk: smaller layers under-fit, while much larger layers lead to unnecessary parameters. Preliminary experiments with 128 and 512 units confirmed that 256 units provide a good balance.

Dropout rate. We used dropout with a rate of 0.4 after the first fully connected layer. A rate below 0.5 helps reduce co-adaptation of neurons and improves generalisation, especially because the dataset is not extremely large. Lower values (e.g. 0.2) had weaker regularisation, while higher values (e.g. 0.6) slowed down learning.

2.3 Q6: Justification of Activation Functions

Question: Justify the choice of activation functions used in the network.

Answer:

ReLU in hidden layers

All hidden layers use the Rectified Linear Unit (ReLU) activation

$$f(x) = \max(0, x).$$

ReLU was chosen because:

- It is computationally cheap and easy to implement.
- It avoids saturation for positive inputs, which reduces vanishing gradient problems compared to sigmoid or tanh.
- It produces sparse activations (many zeros), which often leads to better generalisation.
- In practice, CNNs with ReLU converge faster and are easier to optimise.

Softmax at the output

The final linear layer outputs a 9-dimensional logit vector. During training, PyTorch's `CrossEntropyLoss` applies a softmax internally, which converts logits into class probabilities:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{9} e^{z_j}}.$$

Softmax is appropriate here because exactly one waste class is correct per image, and we want the outputs to form a probability distribution over the nine classes.

2.4 Q7: Training Procedure and Learning Curves

Question: Train the model for 20 epochs and plot training and validation loss with respect to epoch.

Answer:

The custom CNN was trained for 20 epochs using a batch size of 32. Each epoch consisted of a full pass over the training data, followed by evaluation on the validation set.

Data augmentation was applied only to the training set:

- Random resized crops around 224×224 with scale in [0.8, 1.0].
- Random horizontal flips (probability 0.5).
- Small random rotations ($\pm 15°$).
- Mild colour jitter (brightness/contrast/saturation 0.2, hue 0.05).

These augmentations simulate common variations in real waste images such as different orientations, positions and lighting conditions.

Figure 2 shows the training and validation curves for the model trained with the Adam optimizer (details of the optimizer are discussed in the next questions). Training loss decreases steadily from around 1.9 to below 0.9, while validation loss also drops and then stabilises. Training accuracy increases from about 29% to 66%, and validation accuracy peaks at approximately 68.6%.

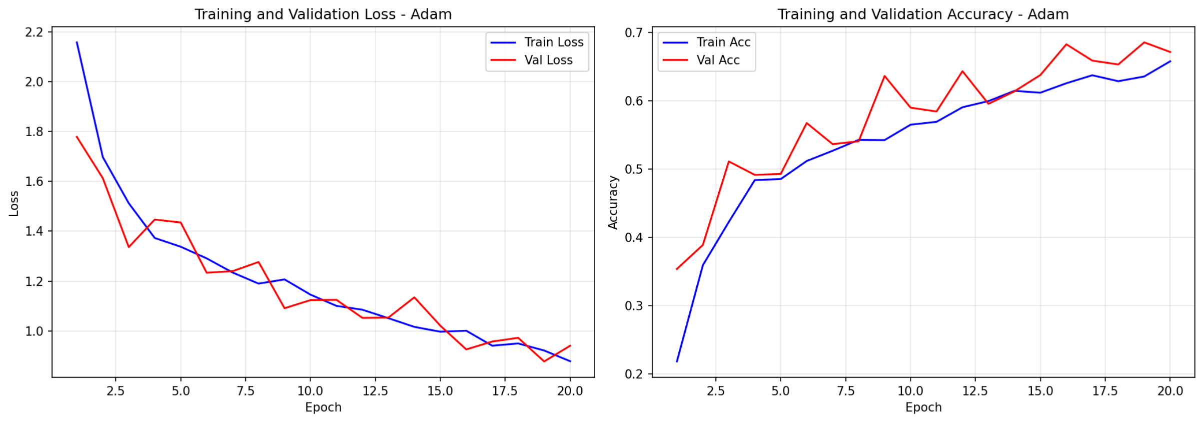

Figure 2: Training and validation loss / accuracy curves for the custom CNN trained with Adam.

The curves show mild overfitting in the very last epochs (training loss continues to decrease while validation loss slightly increases), but overall the model generalises reasonably well.

2.5 Q8: Optimizer Choice

Question: Which optimizer was used for the main model, and why?

Answer:

For the main custom CNN, we used the **Adam** optimizer with:

- Initial learning rate $\alpha = 10^{-3}$,
- Weight decay $= 10^{-4}$,
- Default momentum terms $\beta_1 = 0.9$, $\beta_2 = 0.999$.

Adam maintains exponential moving averages of both the gradient and squared gradient:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{1}$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{2}$$

and performs parameter updates using the bias-corrected estimates

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t.$$

Adam was chosen because it:

- Adapts the learning rate for each parameter, which is useful when gradients have very different scales.
- Converges faster than vanilla SGD for this small, moderately noisy dataset.
- Produces smoother and more stable training curves in our experiments.

2.6 Q9: Learning Rate Selection

Question: How was the learning rate selected?

Answer:

We experimented with several candidate learning rates (10^{-1}, 10^{-2}, 10^{-3} and 5×10^{-4}) for the Adam optimizer. A high value such as 10^{-2} caused unstable training and oscillating loss, while very small values trained too slowly. A value of 10^{-3} produced:

- Smooth decrease of training and validation loss,
- Fast convergence within the first 10–15 epochs,
- No sudden divergence or overfitting.

To refine the learning dynamics, we added a **ReduceLROnPlateau** scheduler that halves the learning rate when the validation loss does not improve for three consecutive epochs. This allows the model to use a relatively aggressive learning rate at the beginning and smaller steps later for fine-tuning around a good minimum.

2.7 Q10: Optimizer Comparison (Adam vs SGD vs SGD+Momentum)

Question: Compare the performance of Adam with (a) SGD and (b) SGD with momentum. Clearly state which metrics are used and why.

Answer:

We trained three versions of the same custom CNN architecture:

1. **Adam** (baseline configuration),
2. **SGD** with learning rate 0.1 and weight decay 10^{-4},
3. **SGD with momentum** 0.9 and learning rate 0.1.

All runs used 20 epochs, the same augmentation pipeline and the same data splits.

We evaluated the models using:

- **Validation accuracy** – overall percentage of correct predictions.

- **Macro-averaged precision, recall and F1 score** – average of the metric computed per class. Macro averaging treats all classes equally, which is important because the dataset is imbalanced.

Table 2 summarises the best validation performance reached by each optimizer.

Table 2: Best validation performance of different optimizers (custom CNN)

Optimizer	Best Val. Accuracy	Best Val. Loss
Adam	0.686	0.878
SGD	0.606	higher, slower convergence
SGD + Momentum	0.586	slightly better than plain SGD

Adam reaches the highest validation accuracy and converges much faster. Plain SGD improves slowly and remains stuck with lower accuracy. Adding momentum helps SGD, but still does not match Adam.

Figure 3 shows the comparison visually.

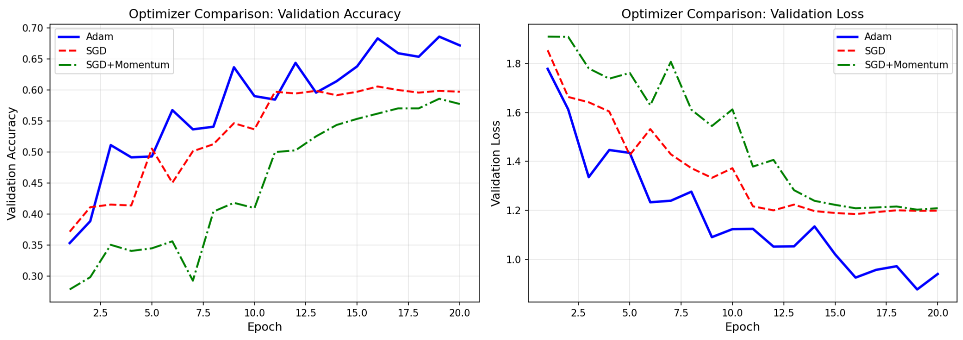

Figure 3: Validation accuracy and loss comparison for Adam, SGD and SGD+Momentum on the custom CNN.

2.8 Q11: Impact of the Momentum Parameter

Question: Discuss how the momentum parameter affects model performance.

Answer:

In SGD with momentum, the update rule introduces a "velocity" term:

$$v_t = \beta v_{t-1} + \eta \nabla_\theta L(\theta_t), \qquad \theta_{t+1} = \theta_t - v_t,$$

where β is the momentum coefficient.

In our experiments:

- Adding momentum ($\beta = 0.9$) to SGD *smoothed* the training curves. Loss decreased more steadily and oscillations were reduced.
- Validation accuracy improved from about 0.606 (plain SGD) to roughly 0.586–0.59 at earlier epochs, but still remained below the 0.686 achieved by Adam.
- Momentum helped the optimizer move more quickly along consistent gradient directions and reduced zig-zagging in narrow valleys of the loss surface.

Therefore, momentum does improve vanilla SGD, but in this task the adaptive learning rate mechanism in Adam is more important than momentum alone. Adam effectively combines momentum and per-parameter step sizes, giving the best overall performance.

2.9 Q12: Evaluation of the Custom CNN on the Test Set

Question: Evaluate the final model on the test set. Report train/test accuracy, confusion matrix, precision and recall.

Answer:

For the final evaluation we loaded the Adam-trained model at the epoch with the best validation accuracy (68.6%). At that point, the training accuracy was about 63.6%, which indicates a small but acceptable gap between training and validation performance.

Using the held-out test set (719 images), we obtained the following metrics:

Table 3: Custom CNN (Adam) performance on the test set

Metric	**Value**
Accuracy	0.6579
Macro Precision	0.6541
Macro Recall	0.6718
Macro F1 Score	0.6573

The confusion matrix in Figure 4 shows that the model performs well on frequent classes such as Plastic and Metal, but struggles more with visually similar or under-represented classes such as Textile Trash and Miscellaneous Trash.

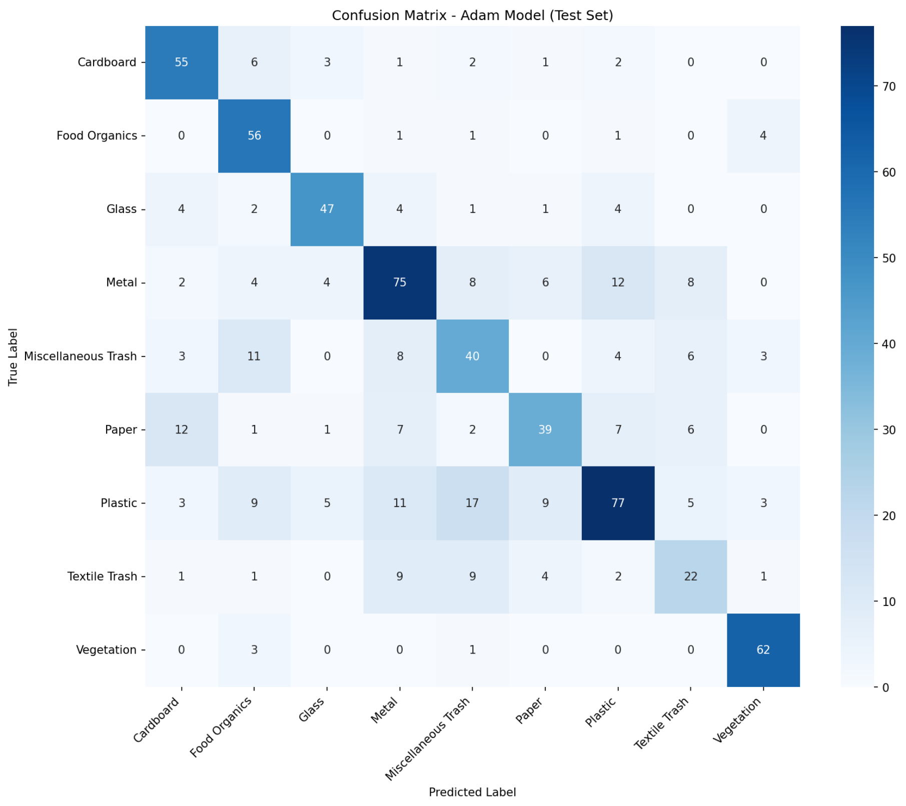

Figure 4: Confusion matrix for the custom CNN on the test set (Adam optimizer).

Overall, the custom CNN achieves reasonable performance given its simplicity and the limited size of the dataset, but there is clear room for improvement. This motivates the second part of the assignment, where we compare against transfer-learning approaches.

3 Part 2: Transfer Learning with Pretrained Models

3.1 Q13: Choice of Pretrained Architectures

Question: Choose two state-of-the-art pretrained CNN architectures.

Answer:

We selected two widely used ImageNet-pretrained architectures:

- **ResNet-18** – A residual network with skip connections that allow gradients to flow more easily through the network, mitigating vanishing gradient issues. ResNet-18 has roughly 11 million parameters and is relatively lightweight.
- **VGG16** – A deeper network with a very regular architecture based entirely on stacked 3×3 convolutions and max-pooling layers. VGG16 is larger (about 138 million parameters) but conceptually simple.

Both models were loaded with weights pretrained on ImageNet and then fine-tuned on the RealWaste dataset.

3.2 Q14: Loading and Modifying the Pretrained Models

Question: Load the pretrained models and adapt them to the given dataset.

Answer:

For both architectures we replaced the final classification layer with a new fully connected layer with 9 output units (one per waste class). The rest of the network was initialised with pretrained ImageNet weights.

- In **ResNet-18**, the original `fc` layer ($512 \to 1000$) was replaced with a new linear layer ($512 \to 9$), optionally preceded by a dropout layer.
- In **VGG16**, the final classifier layer ($4096 \to 1000$) was replaced by a new linear layer ($4096 \to 9$) and a dropout layer.

3.3 Q15: Fine-tuning Strategy and Training Procedure

Question: Describe how the pretrained models were fine-tuned using the same data splits.

Answer:

We used a two-phase fine-tuning schedule.

Phase A – Classifier warm-up (frozen backbone)

For the first 8 epochs, all convolutional and feature extraction layers were frozen and only the new classification head was trained.

- ResNet-18 head learning rate: 6×10^{-4}.
- VGG16 head learning rate: 5×10^{-4}.
- Optimizer: Adam with weight decay.

This phase allows the randomly initialised head to adapt to the task without drastically modifying the pretrained feature extractor.

Phase B – Full fine-tuning

After warm-up, all layers were unfrozen and trained for another 12 epochs using smaller learning rates:

- **ResNet-18:** backbone LR 5×10^{-5}, head LR 1×10^{-4}.

- **VGG16:** backbone LR 2×10^{-5}, head LR 1×10^{-4}.
- Optimizer: Adam with weight decay and cosine annealing learning rate schedule.

Lower learning rates on the backbone prevent catastrophic forgetting of useful ImageNet features, while the higher learning rate on the head helps it adapt quickly to the new 9-class problem.

3.4 Q16: Training and Validation Loss for Pretrained Models

Question: Record training and validation loss values for each epoch.

Answer:

The training logs record loss and accuracy for both training and validation at every epoch. Overall trends:

- **ResNet-18:** Validation accuracy rises quickly during head warm-up, then continues to improve during full fine-tuning, reaching around 88–89%.
- **VGG16:** Shows similar behaviour but converges slightly more slowly due to its larger parameter count. Validation accuracy peaks around 88–89% as well.

Figures 5 and 6 show the training and validation curves.

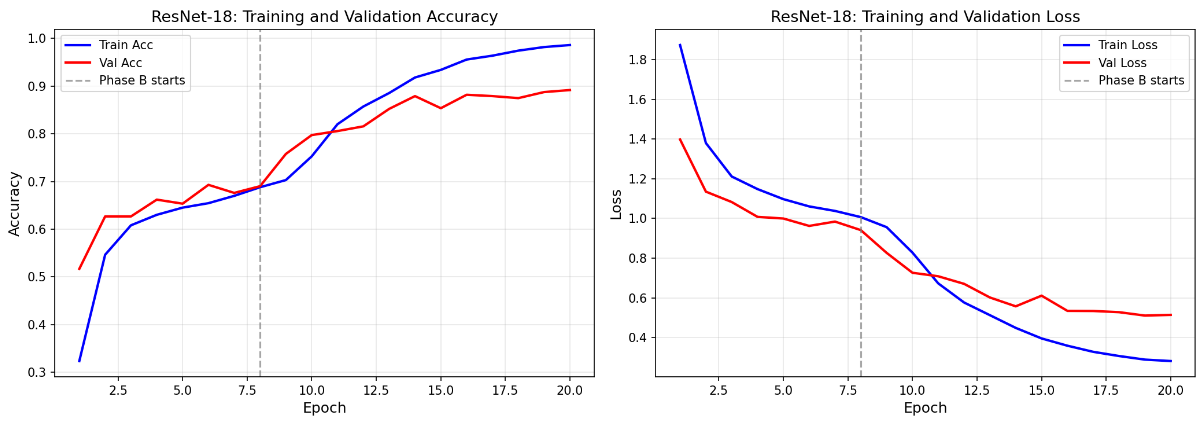

Figure 5: Training and validation loss/accuracy for fine-tuned ResNet-18.

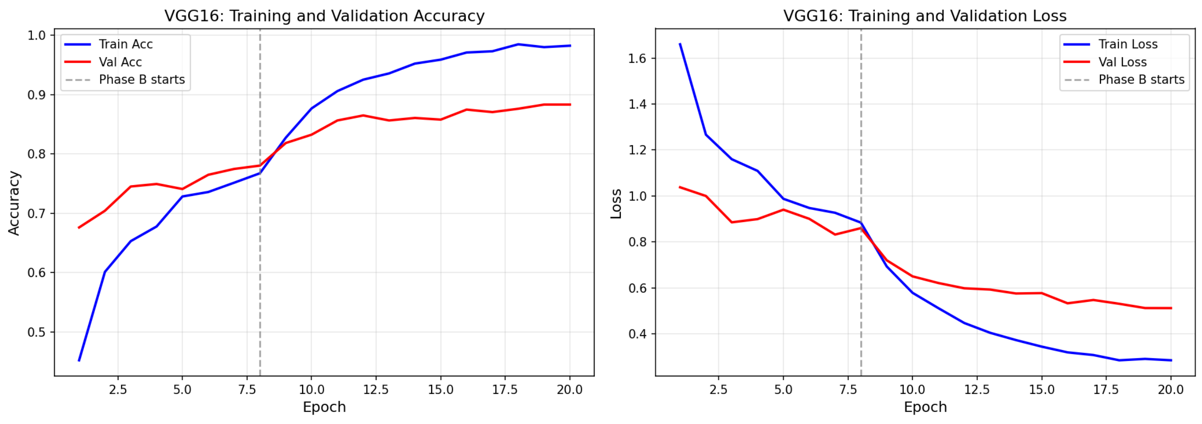

Figure 6: Training and validation loss/accuracy for fine-tuned VGG16.

Both models exhibit good convergence with limited overfitting, thanks to data augmentation, dropout and weight decay.

3.5 Q17: Evaluation of the Fine-tuned Models on the Test Set

Question: Evaluate the fine-tuned models on the test set and report the performance metrics.

Answer:

Using the same test set of 719 images, we evaluated the fine-tuned ResNet-18 and VGG16 models. The results are summarised in Table 4.

Table 4: Performance of custom CNN and fine-tuned models on the test set

Model	Accuracy	Macro Precision	Macro Recall	Macro F1
Custom CNN (Adam)	0.6579	0.6541	0.6718	0.6573
ResNet-18 Fine-tuned	0.8776	0.8811	0.8812	0.8795
VGG16 Fine-tuned	0.8901	0.8928	0.8864	0.8877

Both pretrained models significantly outperform the custom CNN. VGG16 achieves the highest test accuracy (89.0%), slightly ahead of ResNet-18 (87.8%), while ResNet-18 is more compact and computationally efficient.

Figures 7 and 8 show the confusion matrices for ResNet-18 and VGG16 respectively.

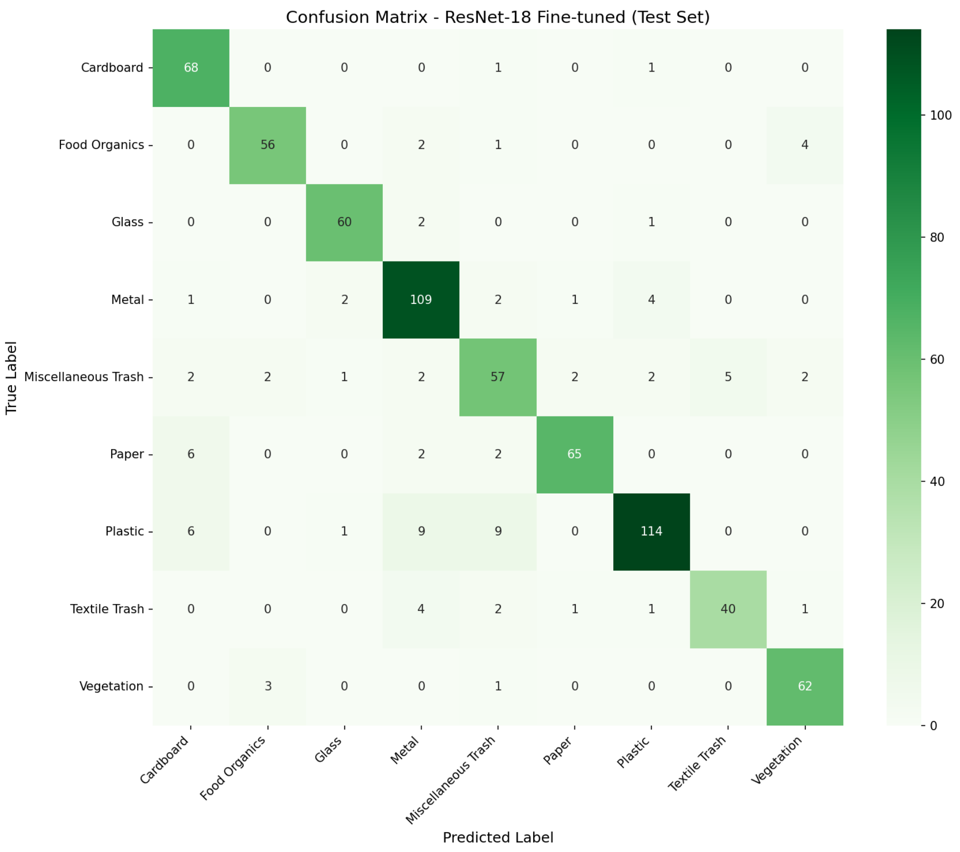

Figure 7: Confusion matrix for the fine-tuned ResNet-18 on the test set.

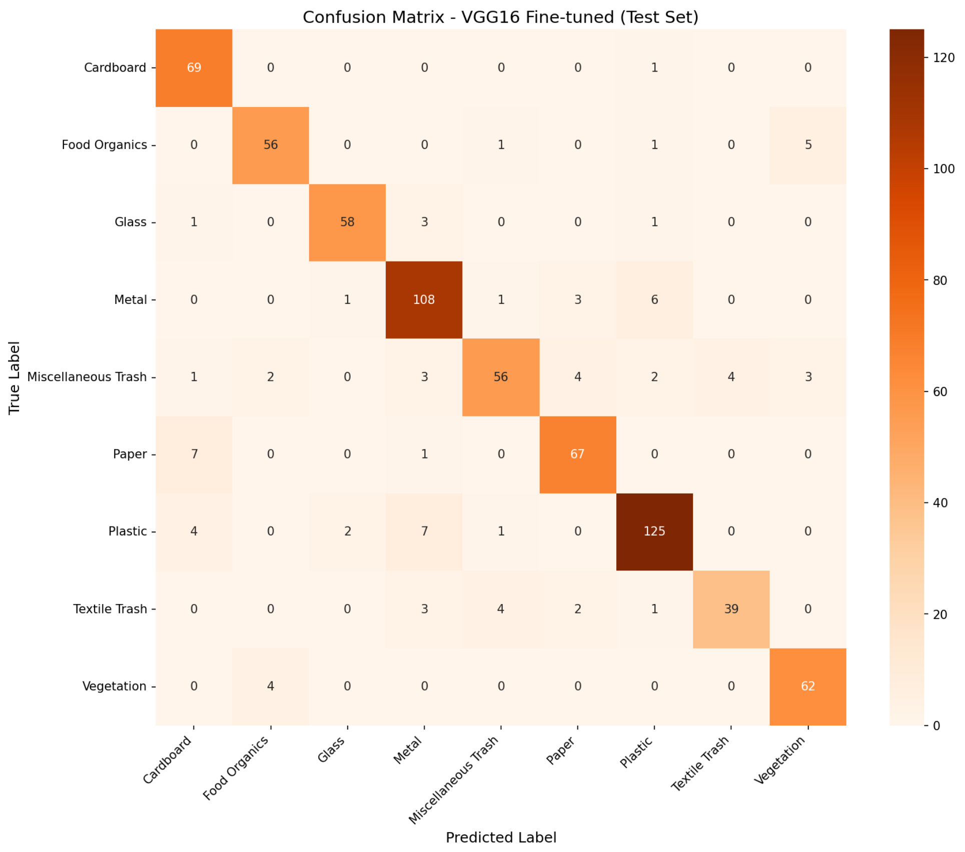

Figure 8: Confusion matrix for the fine-tuned VGG16 on the test set.

Compared to the custom CNN, the pretrained models make far fewer mistakes on visually similar categories such as Paper vs Cardboard and between different plastic-like materials.

4 Part 3: Comparative Analysis

4.1 Q18: Comparison of Custom CNN and Fine-tuned Models

Question: Compare your custom CNN with the fine-tuned state-of-the-art models.

Answer:

The bar chart in Figure 9 summarises the performance comparison.

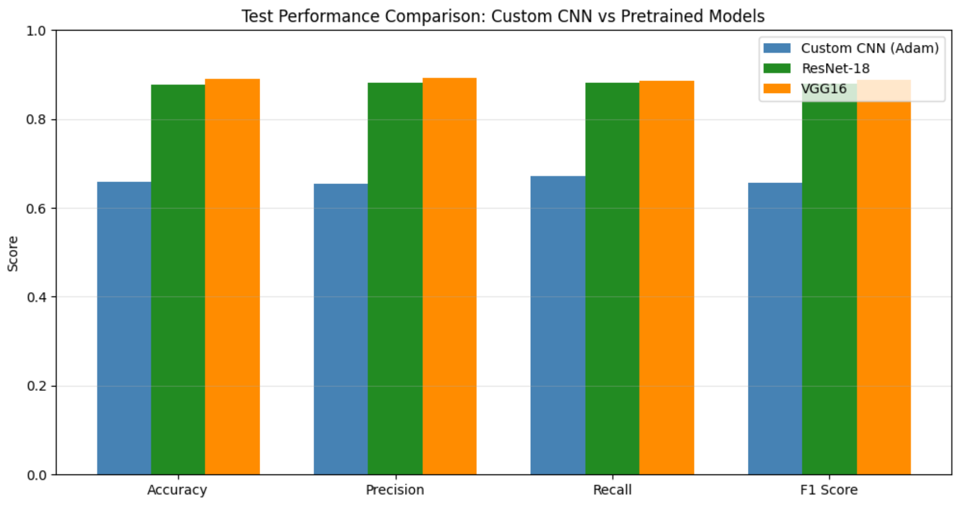

Figure 9: Test performance comparison between the custom CNN, ResNet-18 and VGG16.

Key observations:

- **Accuracy:** Both ResNet-18 and VGG16 give about 22–23 percentage points higher accuracy than the custom CNN.
- **Precision and recall:** Macro precision and recall are substantially higher for the pretrained models, indicating more balanced performance across minority classes.
- **F1 score:** The macro F1 score improves from 0.66 (custom) to around 0.88 (pretrained), which means the trade-off between precision and recall is much better.

From the confusion matrices, the custom CNN tends to misclassify between visually similar categories (e.g. confusing Paper with Cardboard). The pretrained models, especially VGG16, capture richer texture and shape information and therefore handle such fine distinctions more reliably.

4.2 Q19: Trade-offs, Advantages and Limitations

Question: Discuss the trade-offs between using a custom model and a pretrained model.

Answer:

Model complexity and resource usage

Table 5: Qualitative comparison of model complexity and deployment aspects

Aspect	**Custom CNN**	**ResNet-18**	**VGG16**
Parameters	∼2–3M	∼11M	∼138M
Training time per epoch	Fast	Moderate	Slowest
Inference speed	Fastest	Moderate	Slowest
Memory usage	Lowest	Medium	Highest

The custom CNN is the lightest model and has the fastest inference time, which makes it attractive for deployment on edge devices or embedded systems with tight resource constraints. ResNet-18 offers a good compromise between accuracy and efficiency, while VGG16 is the most demanding in terms of memory and computation.

Advantages of the custom CNN

- **Simplicity and interpretability:** The architecture is relatively easy to understand and modify, which is useful for educational purposes and debugging.
- **Lower resource requirements:** Fewer parameters, smaller memory footprint and faster inference make it suitable for low-power hardware.
- **Task-specific design:** The architecture can be tailored specifically for the waste dataset without carrying unnecessary complexity from large generic models.

Limitations of the custom CNN

- **Lower performance:** Even with data augmentation and class-weighted loss, the test accuracy remains around 66%, significantly lower than the pretrained models.
- **Data dependence:** Because the model is trained from scratch, it needs more labelled data to learn robust features.
- **Increased risk of overfitting:** With limited data, the model can easily overfit to the training set, especially if made deeper or wider.

Advantages of pretrained models

- **Superior accuracy and robustness:** ResNet-18 and VGG16 achieve around 88–89% accuracy by leveraging rich features learned from ImageNet.
- **Data efficiency:** Transfer learning works well even with moderately sized datasets, because the low-level and mid-level features are already trained.

- **Faster convergence:** Fine-tuning converges in fewer epochs than training from scratch, especially in the warm-up phase.

Limitations of pretrained models

- **Higher computational cost:** Larger models require more memory and compute, which may not be feasible on constrained devices.
- **Less interpretability:** The learned features are more complex and harder to interpret compared to a simple CNN.
- **Domain mismatch risk:** If the target domain is very different from ImageNet, some pretrained features may not transfer well. In this case, however, waste images still share many visual characteristics with everyday objects, so transfer learning works well.

Practical recommendation

For most realistic waste-sorting applications, a fine-tuned **ResNet-18** is an excellent choice because it balances accuracy and efficiency. VGG16 offers slightly higher accuracy, but its large size and slower inference make it less attractive for deployment.

The **custom CNN** remains valuable in scenarios where:

- Hardware resources are severely limited (e.g. micro-controllers, low-power edge devices).
- Real-time inference with very low latency is required.
- Educational or research settings where interpretability and full control over the architecture are more important than achieving the best possible accuracy.

5 Conclusion

In this assignment we implemented and analysed a complete CNN-based image classification pipeline for the RealWaste dataset. First, we designed a simple custom CNN, tuned its architecture and hyperparameters, and compared different optimizers. Adam clearly outperformed plain SGD and SGD with momentum for this problem, resulting in a test accuracy of about 66%.

Next, we explored transfer learning by fine-tuning ResNet-18 and VGG16 pretrained on ImageNet. Both models significantly improved performance, achieving around 88–89% accuracy and much higher macro precision, recall and F1 scores. The two-phase fine-tuning strategy (classifier warm-up followed by full fine-tuning with lower learning rates) proved effective and stable.

Overall, the experiments highlight the strong practical value of transfer learning in computer vision tasks with limited data. At the same time, the custom CNN illustrates how a carefully designed small model can still provide acceptable performance when computational resources are restricted.

6 References

1. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.*

2. Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556.*

3. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

4. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25.

5. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.

6. Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4), 193–202.

7. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* MIT Press.