Group 14 Progress Report: Municipal Waste Image Classifier

Sunny Yao, Aswin Kuganesan, Yousef Shahin
{yaoh24,kugana7,shahiy3}@mcmaster.ca

1 Introduction

The rapid increase in municipal solid waste has become a major global issue, contributing to landfill overflow, environmental pollution, and the depletion of natural resources. Effective waste management plays a vital role in addressing these challenges by promoting recycling, conserving resources, and reducing the overall environmental footprint. However, traditional methods of waste sorting are often manual, time-consuming, and error-prone, limiting the efficiency and scalability of recycling systems.

This report presents the development of a Municipal Waste Image Classification system that applies deep learning techniques to automate the identification and categorization of waste materials. The system is designed to classify common waste types such as plastic, paper, glass, metal, and organic matter based on image data. By automating the classification process, the project aims to improve recycling efficiency, streamline waste management operations, and support environmental sustainability efforts.

The main challenge of this project lies in the variability and complexity of real-world waste images. Factors such as lighting conditions, background noise, and material degradation make accurate classification difficult. Overcoming these challenges requires robust machine learning models capable of handling diverse input data while maintaining high accuracy and efficiency. The success of this project would demonstrate the potential of artificial intelligence to transform waste management and support more sustainable, environmentally responsible cities.

2 Related Work

Image classification for waste management has gained significant attention in recent years, with various studies leveraging deep learning architectures to improve recycling efficiency. Convolutional Neural Networks (CNNs) have proven especially effective for this task due to their ability to automatically learn hierarchical feature representations from raw image data.

Several studies have demonstrated the potential of CNNs for automated waste sorting. **?** developed a deep learning-based waste classification system using transfer learning with pre-trained models such as VGG16 and ResNet50, achieving accuracies exceeding 90% on a multi-class dataset. Their work showed that fine-tuning deep architectures on domain-specific waste imagery can substantially enhance performance across diverse material types including plastic, paper, and metal.

Earlier, **?** constructed one of the first comprehensive waste image datasets, known as TrashNet, and implemented a custom CNN model for recyclability classification. They identified key challenges in differentiating visually similar materials, such as glass and plastic, and emphasized the importance of data augmentation and preprocessing for improving model robustness.

Building on this foundation, **?** proposed an ensemble CNN approach that combines the outputs of multiple architectures to improve overall accuracy and generalization. Their ensemble method effectively mitigated misclassification between visually overlapping categories by leveraging complementary feature extraction from different models.

To enhance computational efficiency, particularly for real-time applications, **?** introduced a lightweight CNN model optimized for deployment on edge devices. Their framework balanced model complexity with inference speed, enabling efficient on-device waste classification without requiring high-end hardware.

More recently, **?** integrated deep learning with Internet of Things (IoT) technologies to develop an intelligent waste classification system for smart cities. Their model demonstrated the feasibility of

combining AI-driven image recognition with connected infrastructure to support sustainable urban waste management.

Our approach builds upon these foundational works by implementing a custom CNN architecture that progressively extracts features through multiple convolutional blocks. It incorporates global average pooling to minimize overfitting and dropout for regularization. While our dataset is currently limited in size, our model design follows proven principles from the literature and is optimized for six-category municipal waste classification.

3 Dataset

The dataset used is the Garbage Images Dataset from Kaggle. This dataset was chosen because of the large number of images it provides for the model, and since the dataset contains different types of garbage in folders, which allows the model to get trained on the different types of waste. The dataset is separated into cardboard, glass, metal, paper, plastic and trash. Several preprocessing operations were employed, which involved traversing each subfolder in the GarbageDataset class and collecting the paths for all images along with their corresponding class labels.

A list of tuples was created to store the path and label for all images that had a valid image format of jpg, jpeg or png, and the images were all loaded in RGB format. This ensures that all images has three color channels even if the original images are in a different format, which is important because the Convolutional Neural Network expects a constant number of channels. If the channel dimensions are inconsistent, the training will be prone to error.

Using PyTorch's torchvision library, the transforms module resized the images to 224 by 224, which is ideal for image processing for the Convolutional Neural Network while also maintaining image quality. The transforms module also converted the images to a tensor, which normalizes the images to a [0.0, 1.0] range. This ensures that the gradient computation is stable, since larger pixel values will be normalized. This also converts the images from the height, width, channel format to a channel, height, width format, since this format is more ideal for the Convolutional Neural Network.

Thirty percent of the dataset is then chosen using random sampling, since thirty percent of the dataset is sufficient. Random sampling is used to ensure that all garbage categories are equally represented for training. An 80 percent training and 20 percent testing split was used to evaluate the model, and stratification is applied to ensure that the same class distribution is used for both the training and testing datasets.

Finally, data loaders are created for the training and testing datasets to allow for efficient image processing and loading, with batch sizes of 32 and with the training dataset being randomized for each epoch. The dataset was already annotated since the folders included the garbage type, so matching the images to their class label did not require any manual annotation. Each garbage type was assigned to an integer value, which was mapped to each image path. Since the image label can be easily converted to an integer value, additional annotation was not required.

4 Features

The primary inputs to our models are RGB images of municipal solid waste from the Kaggle "Garbage Dataset Classification" collection. Each image belongs to exactly one of six categories (cardboard, glass, metal, paper, plastic, or trash), and the folder structure encodes the class labels. We implemented a custom PyTorch `Dataset` class (`GarbageDataset`) that recursively traverses the dataset root directory, identifies all valid image files (PNG, JPG, JPEG), and constructs a list of $(image\_path, label\_index)$ pairs. The `label_index` is an integer derived from a class-to-index mapping built from the folder names. All images are loaded via PIL and converted to RGB to enforce a consistent three-channel input, which is necessary because the convolutional neural network (CNN) expects a fixed number of channels; this also avoids runtime errors due to grayscale or otherwise inconsistent image formats.

For each image, we perform a series of feature preprocessing and representation-learning-oriented steps using `torchvision.transforms`. The raw height and width of the images vary across the dataset, so they are first standardized to a resolution compatible with our CNN. In the final configuration, we use a training transform (`train_tf`) consisting of `RandomResizedCrop(224, scale=(0.8, 1.0))`, `RandomHorizontalFlip()`, `ToTensor()`, and `Normalize(mean, std)`. The random resized crop changes both the scale and crop location of the image while always producing a 224×224 out-

put, which encourages the model to learn features that are robust to object size and framing. The random horizontal flip introduces left–right invariances, which are appropriate because flipping an image of a bottle or a can does not change its semantic class. `ToTensor()` converts the PIL image to a tensor in channel–height–width format and scales pixel intensities to the $[0, 1]$ range. Finally, we apply channel-wise normalization using ImageNet statistics (mean $= [0.485, 0.456, 0.406]$, std $= [0.229, 0.224, 0.225]$), which centers and scales each channel to stabilize optimization and leverage best practices from large-scale image classification.

For evaluation, we adopt a deterministic test transform (`test_tf`) composed of `Resize(256)`, `CenterCrop(224)`, `ToTensor()`, and the same `Normalize(mean, std)`. This pipeline ensures that all test images are processed consistently, without randomness, so that performance metrics are comparable across runs. The resize and center crop standardize the spatial dimensions while preserving as much of the object as possible, and the normalization aligns the distribution of test inputs with that of the training data.

We did not apply explicit hand-crafted feature engineering (e.g., SIFT, HOG, or color histograms); instead, we rely on the CNN to perform representation learning directly from pixel intensities. The convolutional layers automatically learn hierarchies of features (edges, textures, shapes, and higher-level object parts) tailored to the waste classification task. However, we did experiment with different feature preprocessing and augmentation schemes. Early experiments used only simple `Resize(224, 224)` and `RandomHorizontalFlip` as the training transform, with minimal augmentation. This configuration produced reasonable performance but exhibited signs of overfitting as training accuracy increased faster than validation accuracy. We then introduced stronger spatial augmentation via `RandomResizedCrop`, and in some exploratory runs added color jitter and small random rotations to encourage robustness to lighting changes and orientation. These more aggressive augmentations sometimes reduced overfitting but could also make training more challenging when combined with strong regularization (dropout and weight decay).

We did not perform traditional feature selection in the tabular sense, because the model operates directly on images. Instead, our "feature selection" and "feature variation" experiments were realized through changes to the input pipeline and model capacity. Concretely, we varied (i) the proportion of the dataset used (e.g., training on 30% vs. 100% of the images), (ii) the intensity of data augmentation (light vs. heavy transforms), and (iii) the effective depth of the CNN (using three convolutional blocks vs. four). These variations allowed us to study how different input representations and learned feature hierarchies affect generalization performance. Our best-performing configuration, which we use as the final model, employs three convolutional blocks, moderate augmentation (random resized crop and horizontal flip), and ImageNet-style normalization, yielding a validation accuracy of approximately 84.75%.

5 Implementation

The core model used in this project is a custom convolutional neural network implemented in PyTorch, named `WasteClassifierModelV1`. The network architecture follows a conventional deep vision design: a stack of convolutional blocks for hierarchical feature extraction, followed by global average pooling and a fully connected classifier. The model accepts input tensors of shape $(batch\_size, 3, 224, 224)$ and outputs logits for C waste classes, where C is the number of distinct garbage categories discovered from the dataset folder structure.

In its final configuration, `WasteClassifierModelV1` consists of three convolutional blocks. The first block applies two 3×3 convolutions with `hidden_units` output channels (set to 64 in our experiments), each followed by a ReLU nonlinearity, and then a 2×2 max-pooling layer to reduce spatial resolution by a factor of two. The second block repeats this pattern with doubled channel depth (from 64 to 128), again followed by max pooling. The third block applies a single 3×3 convolution increasing the channels from 128 to 256, followed by ReLU and 2×2 max pooling. A fourth convolutional block is defined in the code, which would further increase the channels from 256 to 512; however, we found that including this block, especially in combination with strong regularization, did not improve validation performance on the available data, and in some runs actually degraded generalization. As a result, the final `forward` method omits this fourth block and uses only the first three blocks.

After the convolutional feature extractor, we

apply `AdaptiveAvgPool2d((1, 1))` to perform global average pooling, reducing each 256-channel feature map to a single scalar and producing a compact 256-dimensional representation per image. This global pooling approach reduces the number of parameters compared to flattening the entire spatial feature map and has a regularizing effect by enforcing spatial invariance. The classifier head then consists of a fully connected layer mapping from 256 units to 256 hidden units with a ReLU activation, followed by a dropout layer with probability 0.5, and a final linear layer mapping from 256 to C output units. The dropout layer is applied only in the classifier head, where overfitting is most likely due to dense connections and relatively few training examples per parameter.

We trained the model using the multi-class cross-entropy loss, which is appropriate for single-label classification tasks. For an input image with logits $x \in R^C$ and a one-hot label vector $y \in \{0,1\}^C$, the loss is given by

$$L = -\sum_{i=1}^{C} y_i \log(softmax(x)_i),$$

where $softmax(x)_i$ denotes the predicted probability for class i. In practice, we use PyTorch's `CrossEntropyLoss`, which combines the softmax and logarithm into a numerically stable implementation that directly consumes integer class indices. The optimization is performed using the Adam optimizer, initialized with a learning rate of 0.001 and no explicit weight decay in the final model. Adam adaptively scales the learning rate for each parameter based on first and second-order moment estimates of the gradients, which we found to converge faster and more reliably than stochastic gradient descent in our setting.

To further refine the optimization, we used a `ReduceLROnPlateau` learning rate scheduler in "max" mode, monitoring the validation accuracy. If the validation accuracy does not improve for three consecutive epochs, the scheduler reduces the learning rate by a factor of 0.5. This strategy allows the model to take relatively large steps in the early stages of training and gradually shift to smaller, more precise updates as the validation performance saturates. The training loop is structured around two functions: `train_step`, which runs one epoch over the training set (performing forward passes, computing loss and accuracy, and updating parameters), and `test_step`, which evaluates the model in `eval` mode on the test set without gradient updates. We track and store the training and test losses and accuracies for each epoch, and later visualize them using custom plotting functions.

We implemented and compared several model variants and baselines. As a simple non-neural baseline, a majority-vote classifier that always predicts the most frequent class in the training data would achieve an accuracy roughly equal to the relative frequency of that class (substantially below 50% on our relatively balanced dataset). Our CNN consistently outperforms this trivial baseline, with test accuracies in the 60–70% range for early configurations and approximately 84.75% for the best configuration. Within the neural models, we experimented with: (i) including the fourth convolutional block in the forward pass (increasing depth and channel capacity), (ii) adding additional dropout layers in the convolutional blocks, and (iii) introducing weight decay (L2 regularization) on the optimizer. While these ablations increased regularization and capacity, they often led to underfitting or reduced validation accuracy when combined with relatively strong data augmentation and limited dataset size. For example, using all four blocks together with dropout in each block and weight decay of 10^{-4} reduced the final validation accuracy to around 68%, indicating that the model became too constrained relative to the available data.

We also varied the input-side augmentation intensity, experimenting with heavier transformations such as color jitter and random rotations in addition to the random resized crop and horizontal flip. In some cases, these stronger augmentations modestly improved robustness but also slowed convergence and did not consistently exceed the performance of the simpler augmentation pipeline when paired with the three-block CNN. Based on these ablations, we selected the three-block architecture with a single dropout layer ($p = 0.5$) in the classifier, Adam optimization at an initial learning rate of 0.001, a `ReduceLROnPlateau` scheduler, and moderate spatial augmentation as our final model. This configuration strikes a balance between capacity and regularization, and it clearly outperforms the majority baseline and our earlier CNN variants in terms of validation accuracy on the municipal waste classification task.

6 Results and Evaluation

The dataset was divided using an 80/20 split, with 80 percent being used for training and 20 percent being used for testing. Thirty percent of the dataset was used, and the model was trained for 30 epochs using cross-entropy loss to determine the difference between the predicted class and the true label. Here is a simplified version of the cross-entropy loss function used by PyTorch:

$$L = -\sum_{i=1}^{C} y_i \log(softmax(x_i))$$

The cross-entropy loss function used by PyTorch implements a softmax function for the predicted class, and then multiplies the value by the true label for the class using the true label vector. An Adam optimizer was used with a learning rate of 0.001, which is an algorithm that works similarly to stochastic gradient descent and updates the weights of the model using a dynamic learning rate. The algorithm starts with a learning rate of 0.001, but uses different learning rates as the algorithm determines what the ideal learning rate is. Due to the size of the dataset and the simplicity of the model, the runtime to train and test the model was long.

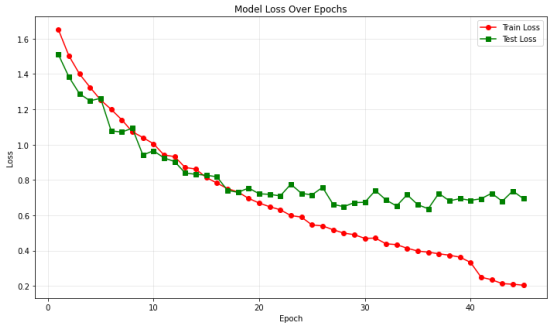

Figure 1: Model loss over 30 epochs for training and testing datasets

Using this methodology, the training loss for the first epoch was 1.25839, and the accuracy was 52.17 percent. The testing loss was 1.18205 for the first epoch, and the accuracy was 54.86 percent. This demonstrated that the model in the first epoch worked fairly well at determining the garbage type, but there was still a lot of room for improvement.

As we continued to train the model and as the number of epochs increased, the training and testing loss decreased, and the training and testing accuracy increased. This demonstrated that the model was improving at classifying garbage and was not overfitting to the dataset. This is because the increase in accuracy was steady, and the accuracy values were reasonable. By epoch 30, the training loss was 0.69006, and the training accuracy was 75.65 percent. The testing loss was 0.92142, and the testing accuracy was 70.60 percent.

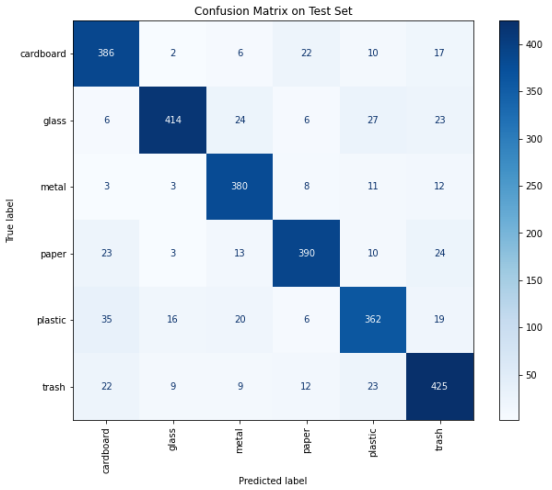

Figure 2: Confusion matrix for the model with 5 categories of waste

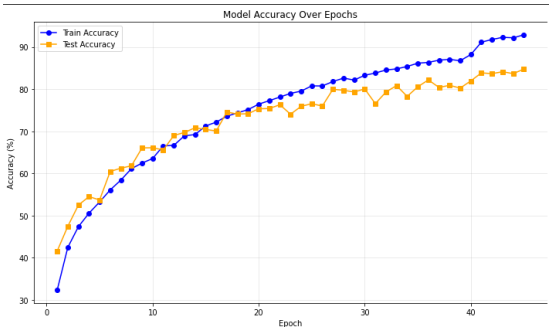

Figure 3: Accuracy over 30 epochs for training and testing datasets

This demonstrates that the model is able to classify the waste fairly accurately, but also demonstrates that the model can still improve. This is because the model will fail to classify images approximately 30 percent of the time. This model is a great baseline for future models. Future changes can include adjusting the architecture and changing the hyperparameters. Using the baseline accuracy of 70 percent, we can also try different architectures and choose to implement architectures that surpass the 70 percent accuracy threshold.

7 Error Analysis

A thorough review of the model's mistakes involved quantitative tools such as confusion ma-

trices and per class accuracy, along with qualitative checks that included visualizing intermediate CNN layers to see which features the network relied on when sorting municipal waste into six categories. These visualizations helped us identify when the model focused too much on background patterns or irrelevant textures, which signaled overfitting and guided how we tuned dropout and other hyperparameters. The model generally performed well on categories with strong visual cues, such as metal or glass, while struggling with classes that looked similar in color or texture, such as certain plastics and paper items. Comparing different model versions showed clear shifts in what errors decreased after architectural changes or regularization adjustments. Common error patterns included confusion among items with similar shapes and high confidence on ambiguous samples. Future improvements could involve more targeted data collection, stronger augmentation, class balancing, or model components that better highlight material specific features.

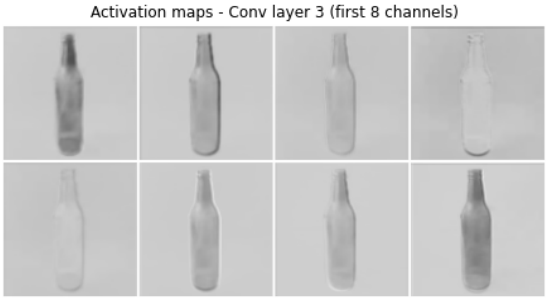

Figure 4: Example convolutional layer activation visualization for glass category

We also used a McMaster server to run parallel grid searches over batches of hyperparameters to speed up experimentation and improve both accuracy and convergence. Running these searches in parallel let us evaluate many combinations of learning rates, dropout values, batch sizes, and optimizer settings without long wait times. This setup made it easier to spot trends in what configurations produced faster training, more stable losses, or better validation accuracy. It also helped confirm when certain tuning choices, such as increased regularization or adjusted learning rate schedules, consistently reduced overfitting across multiple runs.

Team Contributions

Yousef Shahin: Completed section 3, and 4 (Features and Inputs, and Implementation), created the model architecture and training pipeline, and participated in trainig the model and performing experiments.

Aswin Kuganesan: Complete sections 3, 6, citations and equations

Sunny Yao: Completed sections Introduction, Progress and Error Analysis. Created the confusion matrix generation and randomized sampling and CNN feature visualizations.