TÉCNICO LISBOA

Intelligent Systems

Final Project

Automated Deep Learning Waste Classification

Author:

Rita Barco (100259) rita.barco@tecnico.ulisboa.pt
Isabella Silva (100195) isabella.silva@tecnico.ulisboa.pt

2025/2026 – 1st Quarter

Contents

1 Introduction

The growing production of waste has made it increasingly important to have automated systems that can sort and classify garbage efficiently. Common waste-sorting methods rely on manual labor that is slow, prone to mistakes and costly. With the advancement of deep learning, it is now possible to automatically classify waste using image-based models, making the process faster.

This project focuses on building and comparing deep learning architectures for garbage classification, with the goal of designing an intelligent system that can accurately identifying material types from images. Two main models are implemented: a Multilayer Perceptron (MLP) and a Convolutional Neural Network (CNN). The MLP operates on features extracted from an encoder, which was obtained by first training an autoencoder on the dataset and then using its encoder part to generate meaningful feature representations. On the other hand, the CNN learns directly from raw image data. Both models are trained, evaluated and compares to analyze their performance, strenghts and limitations in image-based classification. Evaluation is performed on both studio-like and real-world-like image sets, with performance measured using accuracy, precision, recall, F1 score and confusion matrices.

This project was developed using Pytorch library and can be found on Github.

2 Dataset

The dataset used in this project consists of labeled images of various garbage materials. Each image belongs to a specific class, the classes being: battery, biological, cardboard, clothes, glass, metal, paper, plastic, shoes, trash. The dataset was randomly divided into training and testing sets with an 80/20 ratio for both implementations. The testing sets were manually categorized into two distinct groups: studio-like and real-world-like images. Studio-like images were defined by their monochromatic, uniform backgrounds, typically resembling controlled photographic environments. In contrast, real-world-like images have domestic or naturally varied backgrounds, reflecting everyday settings and less controlled conditions. The dataset can be found in the Kaggle: Garbage Dataset.

3 MLP implementation

This method involves employing a multilayer perceptron (MLP) for image classification. Its inputs consisted of the flattened encoder outputs from a previously trained autoencoder, that way reducing dimensionality. This flattened data is then standardized before being given to the MLP.

3.1 Data preprocessing

The data preprocessing of this section consists in resizing all the images to 128x128 pixels followed by turning them into a tensor suitable for processing by the neural network (trained encoder). Afterwards, before training, the extracted feature data (train_features.npy) and corresponding labels (train_labels.npy) were loaded and standardized to ensure zero mean and unit variance across features. The data was then converted into PyTorch tensors and wrapped into a TensorDataset, which was fed into a DataLoader with a batch size of 32 to enable mini-batch gradient descent.

3.2 Autoencoder

In this section, a convolutional autoencoder (CAE) was designed based to learn compact and meaningful representations of image data. The CAE implemented is based on the version provided on the course webpage, which was adapted to suit the data.

The developed code is organized into three main parts: the model definition (autoencoder_classes.py), created to facilitate feature extraction, the training pipeline (train_autoencoder.ipynb) and the feature extraction phase (use_encoder.ipynb). Together, these components form a learning framework, where raw images are compressed into low-dimensional representations and then reconstructed, allowing the model to learn essential features.

The encoder acts as the compression mechanism of the system, transforming high-dimensional images into smaller, compact latent representations. It has two convolutional layers (that extract local spatial features such as edges, corners and textures), each followed by a rectified linear unit (ReLU) activation and a max-pooling operation which progressively reduces the spatial dimensions. The decoder reconstructs the original images from the compressed latent space using transposed convolutional layers that upsample the feature maps back to the original image size. Nonlinear activations are applied to introduce flexibility during reconstruction. A sigmoid activation in the output layer ensures that the pixel values remain within the normalized range of 0 to 1. The decoder aims to reproduce the original image as accurately as possible by minimizing the mean squared error (MSE) between the input and the reconstructed image.

The training process begins with data loading and preprocessing, where images are retrieved from the dataset directory, resized to a uniform resolution and converted into tensors suitable for processing by the neural network. The training loop is designed to iteratively update the model parameters using the Adam optimization algorithm, guided by the MSE loss function. Over several epochs, the network learns to compress and reconstruct the images by adjusting its convolutional filters to minimize reconstruction errors. Throughout training, the loss value is monitored to ensure convergence. In Figures 1 and 2, a side-by-side comparison of original and reconstructed images is presented. The first being a studio-like image and the second on a real-world one. These images allows a qualitative assessment of the model's performance.

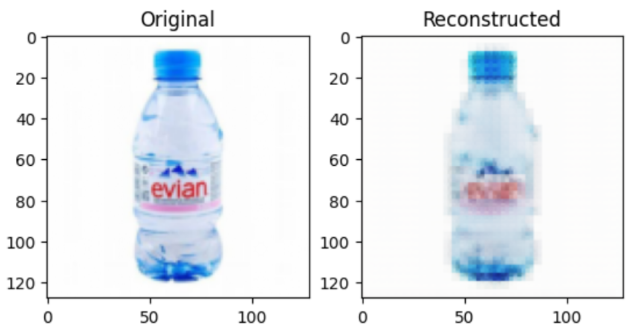


Figure 1: Image reconstruction - Studio-like image example

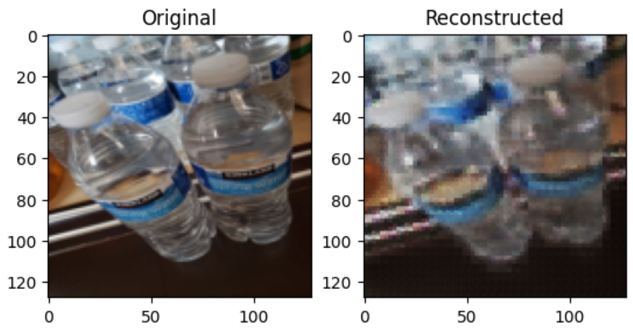


Figure 2: Image reconstruction - Real-world-like image example

Finally, the latent representations produced by the trained encoder (with dimensions corresponding to channels, height, width) are flattened into a one-dimensional format, resukting in a compact feature vector for each image. These feature vectors are then saved as NumPy files, which are later used as inputs for the MLP.

3.3 MLP architecture

Two multilayer perceptron architectures were implemented to classify the features extracted by the autoencoder: a simple baseline model (MLP_Simple) and a deeper model (MLP_Improved).

The MLP_Simple model consists of a single hidden layer with 128 neurons followed by a ReLU activation function and a dropout layer with a probability of 0.1 to mitigate overfitting. The final output layer maps the hidden representation to the number of classes in the dataset using a fully connected (linear) transformation. This configuration provides a lightweight model that is easy to train but has limited capacity to capture complex relationships in the data.

The MLP_Improved model extends on the previous model by introducing additional layers and a hierarchical structure to reduce the feature dimensionality progressively. It begins with a linear layer that reduces the high-dimensional autoencoder features to a lower-dimensional representation (2048 units), followed by two hidden layers with 1024 and 512 neurons, respectively. Each layer is followed by a ReLU activation and dropout with a rate of 0.2, applied to improve generalization and prevent co-adaptation of neurons. The final output layer maps the learned features to class probabilities. This deeper configuration enhances the network's representational power and allows it to capture more complex patterns within the data.

3.4 Training process

Both MLP models were trained using the Adam optimizer with a learning rate of 0.0005 and the Cross-Entropy Loss function, suitable for multi-class classification tasks. The training was carried out for 100 epochs, with loss values printed after each epoch to monitor convergence. During each iteration, the model computed predictions for each batch, compared them to the true labels using the loss function and updated its parameters through backpropagation. Training loss decreased steadily over epochs, indicating successful learning and optimization.

3.5 Testing

After training, the models were evaluated on the two testing datasets (studio-like and real-world-like images). The trained model was switched to evaluation mode (model.eval()), which deactivates dropout layers and disables gradient computations for efficiency. Predictions were generated by performing a forward pass through the network and selecting the class with the highest probability for each input using torch.argmax.

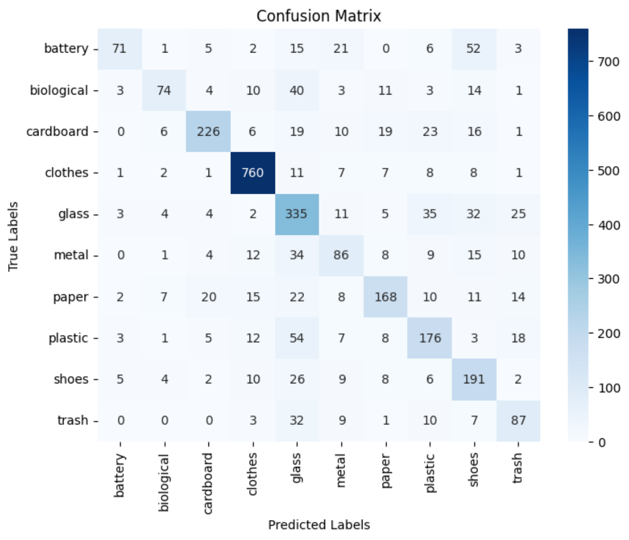
Performance metrics such as accuracy(the proportion of correctly classified examples), precision(the proportion of correctly predicted positive samples among all samples predicted as positive), recall (the proportion of correctly predicted positive samples among all samples predicted as positive) and F1-score (the harmonic mean of precision and recall) were calculated. The MLP_Improved model was chosen as the final MLP model, for it demonstrated better generalization, as can be seen in Table 1 and Figure 3.

The following abreviations were needed to present the results in Table 1:

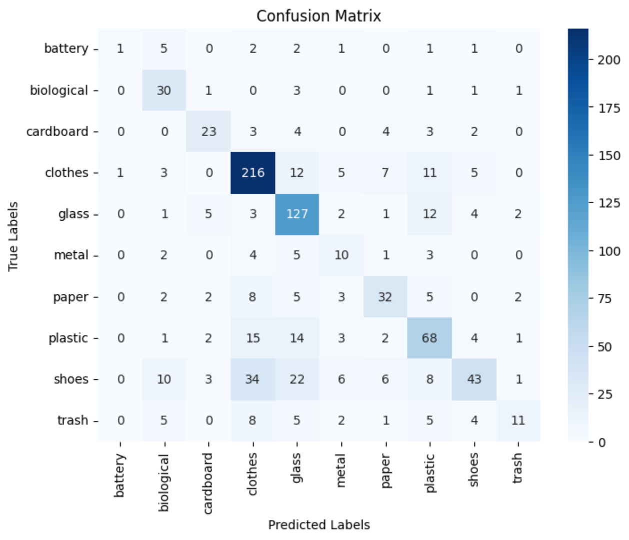
- MLP(S): ML_Simple
- MLP(I): ML_Improved
- T: training dataset
- TeS: test studio-like images
- TeR: test real-world-like images

	MLP (S) T	MLP (I) T	MLP (S) TeS	MLP (I) TeS	MLP(S) TeR	MLP (I) TeR
Accuracy	0.9789	0.9477	0.6862	0.7054	0.6247	0.6419
Precision	0.9791	0.9503	0.6942	0.7200	0.6595	0.6444
Recall	0.9789	0.9477	0.6862	0.7054	0.6247	0.6419
F1 Score	0.9789	0.9473	0.6876	0.7032	0.6213	0.6240

Table 1: Simple and Improved MLP - Results



(a) Studio-like



(b) Real-world-like

Figure 3: Confusion Matrices obtained for MLP_Improved

The results showed that while the MLP_Improved achieved high training accuracy (around 95%), it demonstrated moderate generalization performance on unseen datasets (approximately 71% on the studio-like test set and 67% on the real-world test set), suggesting mild overfitting. The inclusion of dropout and careful tuning of network depth mitigated this issue to some extent. The lower performance on the real-world dataset was expected because these images are more varied and less controlled than the studio-like ones, making them harder for the model to classify.

Additionally, it can be concluded by analyzing the confusion matrices, that the class with the highest classification in both testing scenarios is "clothes" and the lowest is "battery", which might be due to the amount of available images. Classes with larger datasets tend to yield better classification results, reinforcing the importance of balanced data representation. In the studio-like dataset, the confusion matrix reveals several notable misclassifications. The highest number of incorrect predictions involved 52 battery images labeled as shoes, 40 biological images mislabeled as glass, 54 plastic images also classified as glass and 35 glass images misclassified as plastic. These results suggest overlapping visual features between certain material categories, which may have led the model to confuse visually similar textures and colors. Similarly, in the real-world–like dataset, the confusion matrix also indicates some challenges. The most frequent misclassifications included 34 shoes images labeled as clothes, 15 plastic as clothes, 14 plastic as glass and 22 shoes as glass. These errors further highlight how visual similarities (particularly between shoes and clothes or between plastic and glass) can contribute to reduced classification accuracy across datasets.

4 CNN implementation

In this section, a CNN is applied to the task of supervised image classification—given an RGB image, the model predicts one of the predefined waste categories.

4.1 Data preprocessing

Every image is resized to 128×128 pixels and turned into a tensor with 3 channels (RGB).

4.2 CNN architecture

Each sample starts as a tensor of shape [3, 128, 128]. These tensors are then processed through three convolutional blocks, each using 3×3 kernels with padding=1 so feature maps keep their width/height (spatial dimensions remain the same). Channel depth expands (3→8→64→128) through each convolutional layer. After each convolution, ReLU introduces nonlinearity, letting the model encode richer features. Each block ends with 2×2 max pooling, which halves spatial resolution (128→64→32→16), allowing deeper filters to integrate broader context while reducing computation. After the last block, the feature map has shape [128, 16, 16], it is flattened to a vector of length 32768 and passed through a fully connected layer with ReLU, which compresses information from all channels and positions into a 128-value summary. Dropout is applied to this vector to reduce overfitting. Finally, this summary is passed to a second fully connected layer, which produces 1 class scores per category, 10 in total, (logits). The highest score determines the predicted class. In this architecture, an explicit softmax layer is omitted because the loss funtion applied (nn.CrossEntropyLoss) internally applies LogSoftmax, operating directly on raw logits to compute class probabilities during training.

4.3 Training process

The model is trained for supervised classification using the cross-entropy loss function, which compares the predicted logits to the ground-truth class index and penalizes confident mistakes more strongly. Training is done for 50 epochs using the Adam optimizer with a learning rate of 0.001. Adam adaptively scales parameter updates based on estimates of the first and second moments of the gradients, typically resulting in faster and more stable convergence compared to standard stochastic gradient descent (SGD). During each training iteration, a mini-batch of size 32 is passed through the network to perform a forward pass, compute the loss, reset accumulated gradients and execute backpropagation to obtain the gradient of the loss with respect to the model parameters. The optimizer then applies the corresponding parameter updates. Throughout training, the model is maintained in train() mode to ensure dropout is active (0.5). For evaluation, the model is switched to eval() mode and inference is performed within a torch.no_grad() context to disable both dropout and gradient tracking.

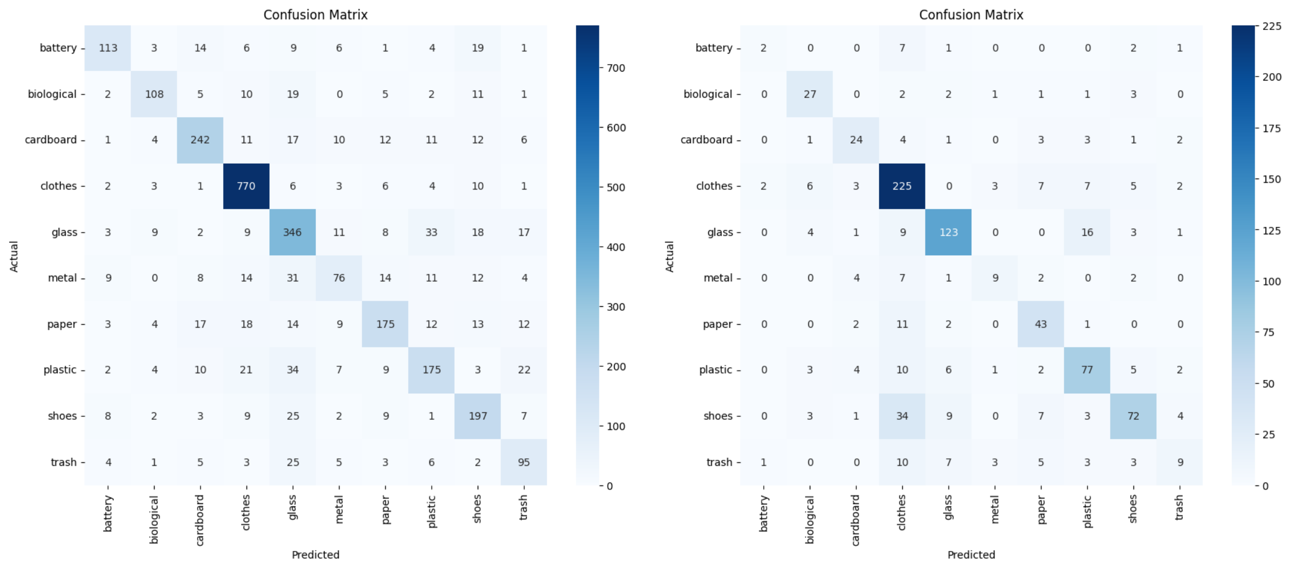
4.4 Testing

Table 2 presents the results obtained for the developed and trained CNN.

	Train	TeS	TeR
Accuracy	0.9970	0.7453	0.6991
Precision	0.9970	0.7436	0.6927
Recall	0.9970	0.7453	0.6991
F1 Score	0.9970	0.7414	0.6875

Table 2: CNN Results

From Table 2, it can be observed that the CNN learned the training set very well, since the training perfomance is almost perfect. However, the performance on both the studio-like test set and the real-world test set drops significantly, clearly indicating overfitting. Furthermore, the metrics on the real-world dataset are worse than those on the studio-like dataset, which is expected because real-world images tend to be noisier and reflect less controlled conditions.



(a) Studio-like **(b)** Real-world-like

Figure 4: Confusion Matrices obtained for CNN

As seen in Figure 4, for the CNN model on the studio.like dataset, the confusion matrix has several misclassifications. The most frequent errors include 33 glass images classified as plastic, 31 metal images as glass and 34 plastic images also classified as glass. COnsidering the real-world-like test set results, the confusion matrix shows that the highest misclassifications were 34 shoes labeled as clothes and 16 glass images classified as plastic.

These results clearly indicated that the CNN faces challenges when categories have overlapping visual features, such as shoes and clothes or glass and plastic.

5 Comparison

The CNN outperforms the MLP_Improved on both test sets, while both models have very high training scores. On studio-like images, the CNN reaches accuracy 0.7453 / F1 0.7414 versus the MLP_Improved's Accuracy 0.7054 / F1 0.7032. On real-world-like images, the CNN attains accuracy 0.6991 / F1 0.6875 compared to accuracy 0.6419 / F1 0.6240 for MLP_Improved. However, the CNN's training performance is almost perfect (Accuracy 0.9970), signaling stronger overfitting than the MLP_Improved (Accuracy 0.9477).

Both models generalize worse to real-world photos than to studio-like ones, but the gap is most pronounced for the CNN and more moderate for the MLP_Improved . This suggests the CNN learned dataset-specific cues very well but still benefited from its spatial inductive bias enough to compensate on both test splits. Analysis of the confusion matrices indicates that performance is poor for the 'battery' class, whereas 'clothes' is the most accurately recognized class in both implementations. Both models are affected by class imbalance and visual similarity (glass with plastic, shoes with clothes), but CNN handles it better, outperfoming the MLP relative to the true positives.

6 Conclusion

This project compared an MLP fed by encoder features with a CNN for waste classification. The CNN achieved the strongest test results on both studio-like and real-world images, though it clearly overfit with near-perfect training scores, while the MLP_Improved was worse on both test sets but overfit less.

Both models degraded from studio to real-world data, confirming a domain shift that limits generalization. Confusion matrices showed systematic mixing between visually similar classes (notably glass versus plastic and shoes versus clothes) and a consistent weakness on battery, whereas clothes was easiest, reflecting data imbalance and overlapping visual cues.

Overall, the CNN is the preferable baseline due to higher test performance and future iterations should focus on closing the generalization gap. This could be done using data augmentation.