Garbage Classification Using Deep Learning

Diogo Marto NºMec 108298
DETI, Universidade de Aveiro
Aveiro, Portugal
diogo.marto@ua.pt

Inês Matos NºMec 124349
DETI, Universidade de Aveiro
Aveiro, Portugal
inessmatos@ua.pt

***Abstract*—This project aims to evaluate and compare the performance of advanced deep learning techniques in the task of automatic classification of household waste through images. Two main approaches were explored: a custom convolutional network (CNN) developed from scratch and a transfer learning-based approach. After partial fine-tuning, the results demonstrated that the transfer learning-based model presented the best overall performance, highlighting the advantages of using weights pre-trained on large datasets for specific tasks with limited datasets. However, the custom CNN also achieved competitive results, confirming its validity as a viable alternative in situations with more restricted computational resources. This work also suggests possible future lines of research to optimize the performance of networks in visually challenging tasks, such as distinguishing similar materials.**

***Index Terms*—Convolutional Neural Network (CNN), Transfer Learning, ResNet50, Images Classification**

This report was made for the FAA course with Petia Georgieva (petia@ua.pt) as its instructor.

I. Introduction

The escalating volume of waste generated globally presents a significant and multifaceted challenge to environmental sustainability and public health. The consequences of improper garbage disposal are far-reaching, contaminating land, air, and water sources, thereby posing substantial threats to both human well-being and the health of the environment. Effective waste management is therefore paramount, and at its core lies the critical process of garbage categorization. Traditionally, this task has been performed manually, but such methods are increasingly recognized as time-inefficient, susceptible to errors, and requiring extensive labor. The surge in waste production, driven by factors such as rising living standards and rapid urbanization, further exacerbates the limitations of manual sorting, underscoring the urgent need for more efficient and accurate solutions.

Accurate classification of waste materials is not merely a preliminary step in waste management; it is a cornerstone for enhancing recycling processes and promoting environmental sustainability. By precisely categorizing waste into specific types, such as glass, paper, plastic, metal, organic matter, and others, it becomes possible to apply appropriate recycling techniques, thereby maximizing resource recovery and minimizing the amount of waste sent to landfills. The ability to classify household garbage into a broader range of categories, beyond the limited two to six classes found in many datasets, allows for a more refined recycling process and has the potential to significantly improve overall recycling rates. The inherent inefficiencies and inaccuracies of traditional waste management methods, which often rely on manual sorting, highlight the necessity for automated systems capable of handling the increasing volume and complexity of waste in a more sustainable and resource-efficient manner.

II. State of the Art

The issue of waste classification has been widely addressed due to the large amount of waste produced and its environmental effects, such as pollution and public health risks [1]. Landfills, for example, not only consume physical space but also contribute to atmospheric, hydrospheric, and geospheric pollution, with waste incineration emitting hazardous compounds into the air [2].

Accordingly, the automatic classification of waste types plays a vital role in minimizing environmental and social impact. To better understand the context, we reviewed recent studies and deep learning architectures used for similar classification problems.

Several works applied Convolutional Neural Networks (CNNs), MobileNet, and ResNet for this task. For instance, the authors in [1] proposed a CNN-based garbage classification algorithm to distinguish between organic, recyclable, and non-recyclable waste, achieving 95% accuracy. Similarly, the work in [3] leveraged transfer learning with InceptionV3 and implemented data augmentation, reaching an accuracy of 92.87%.

Some studies explored alternative strategies. In [4], the DeepWaste application used ResNet50 to perform real-time waste classification on mobile devices, obtaining 88.1% accuracy with real-world images. The comparative analysis in [5] tested AlexNet, VGG16, GoogLeNet, and ResNet using a dataset with 2527 images across six categories. They employed transfer learning and evaluated models using both Softmax and SVM classifiers, with ResNet + SVM achieving 97.86% accuracy.

Another study [6] assessed various models, including simple CNNs, AlexNet, ResNet50 combined with SVM, and Transformers. The results highlighted the superior performance of more advanced models, particularly ResNet50+SVM and Transformer-based approaches, which achieved accuracies around 95%.

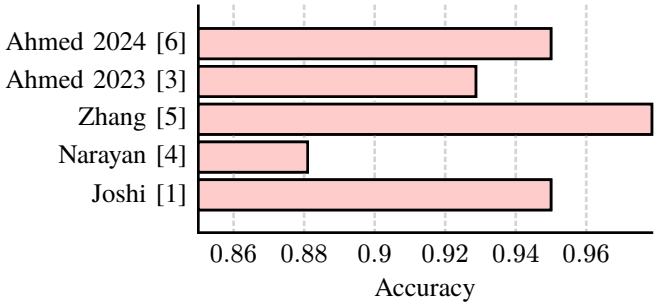


Fig. 1: Barchart of accuracy per model (State of the Art)

In summary, most state-of-the-art approaches employ CNN-based architectures and benefit from small, well-defined datasets such as TrashNet. Although the results are promising, many studies lack deeper performance evaluations and omit techniques like data augmentation or transfer learning. This work aims to address these gaps by evaluating and comparing different models to enhance accuracy and support intelligent waste classification systems.

III. Data Analysis

A. Data Description

The dataset used in this work is related to waste classification and is sourced from the Kaggle platform [7]. It consists of labeled images grouped into six categories:

1) cardboard,
2) glass,
3) metal,
4) paper,
5) plastic,
6) trash.

In total, the dataset contains approximately 2,500 images with a resolution of 512x384 pixels, with each category represented by roughly 400 to 500 samples, except for the trash class, which is significantly underrepresented.

Despite differences in image quality, most images are moderately detailed, as typically seen in lightweight image classification datasets. Sample images for each class are shown in Fig. 2.

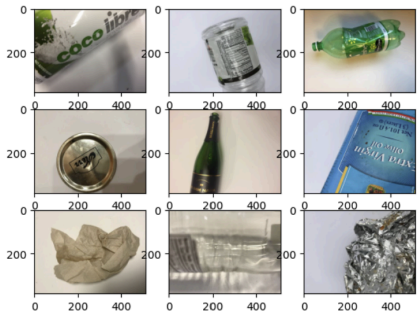


Fig. 2: Images per class in the dataset

B. Data Visualization

Fig. 3 constructs a graphical visualization of the distribution of images by class, from which it is possible to observe that the number of images is not uniformly distributed among the different categories of waste (cardboard, glass, metal, paper, plastic, trash). Furthermore, this disproportionality may introduce challenges in model training, such as bias towards classes with more examples.

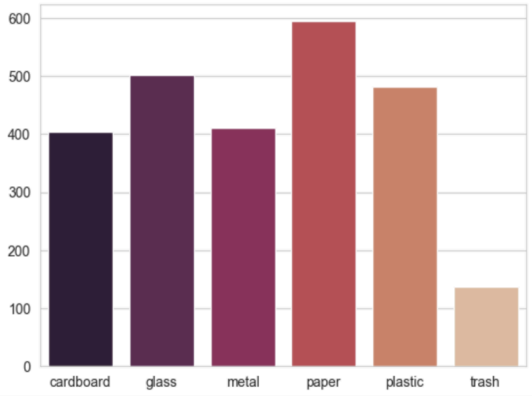


Fig. 3: Number of images per class in the dataset

Early identification of this imbalance is essential, as it may justify using techniques such as oversampling, undersampling, or class weighting during model training. Furthermore, it can also motivate using other classification metrics besides accuracy, such as F1-score, to measure model performance.

In addition, samples of images belonging to various classes were also viewed, as shown in Fig. 4. This visualization allowed us to conclude the visual diversity within the same class, the presence of noise in the images, such as complex backgrounds or partially visible objects, and some differences in lighting and capture angle. In other words, it was possible to identify some details that could impact the generalization capacity of the models.

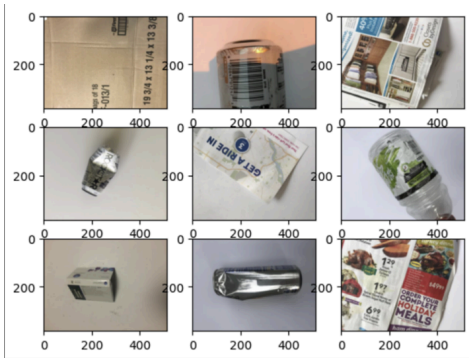


Fig. 4: Samples of Images

C. Data Preprocessing

After analyzing the dataset in question, applying a set of data preprocessing steps was essential to guarantee data homogeneity. Data preprocessing was performed using Keras's *ImageDataGenerator* to prepare the image datasets for model training and evaluation. For the training set, an instance of *ImageDataGenerator* was configured with various transformations including rescaling pixel values to the range [0,1], zooming (up to 10%), rotation (up to 30 degrees), width and

height shifts (up to 20%), and horizontal and vertical flipping. This augmented data generator was used to increase the diversity of the training data and improve model generalization. For validation and testing, a separate *ImageDataGenerator* instance, implemented in the generator function, was used solely for rescaling the pixel values, without applying any geometric or intensity transformations. Another aspect of this approach is the flexibility to specify different *target_size* values, allowing for easy adaptation of the input dimensions to suit the requirements of various trained models. For example, some models might use the full resolution of 512x384 pixels while others can use only 256x192 pixels. In this case, approximately 80% (2021 examples) of the data was split into training and 20% into testing (506 examples), a common proportion given a dataset of this size.

IV. Methods

This work addresses custom convolutional networks developed from scratch and established deep architectures for image classification. For each category, architectural variations, regularization techniques, and training strategies were also tested. In these cases, the *Tensorflow* library was used to implement these models. This made it possible to access GPU optimizations.

A. CNN

Initially, a customized convolutional network (CNN) was implemented, and the base structure was adjusted incrementally so that the impact of each technique on overall performance could be observed. The choice of using CNNs was motivated by the fact that these are suitable for images since, unlike fully connected networks, they preserve the spatial relationships between pixels. Furthermore, 5 are three main types of layers in our architecture, namely convolutional layers (*Conv2D*), pooling layers (*MaxPooling2D*), dense layers (*Dense*), batch normalization layers (*BatchNormalization*), and dropout layers (*Dropout*). Regarding *Conv2D*, these layers extract local features from the image through filters that move over the original image. In turn, *MaxPooling2D* are layers that reduce the dimensionality of the activation maps, preserving the most relevant features; finally, *Dense* are layers used in the final stages of the model to combine the extracted features and perform the classification. A *Batch Normalization* layer standardizes the inputs of a neural network layer by re-centering and re-scaling them to have zero mean and unit variance within each mini-batch, which stabilizes and accelerates training. A *Dropout* layer randomly deactivates a subset of neurons during training, which helps prevent overfitting by reducing complex co-adaptations among neurons.

a) *Base CNN 1:*

An initial architecture was defined, which served as a baseline for subsequent tests, consisting of 5 blocks, each consisting of a convolutional layer and a max pooling layer, then a flattened layer, a dense layer with *ReLU* activation, and a final dense layer with softmax activation, as shown in Fig. 5.

Layer (type)	Output Shape	Param #
batch_normalization (BatchNormalization)	(None, 100, 100, 3)	12
conv2d (Conv2D)	(None, 98, 98, 64)	1,792
max_pooling2d (MaxPooling2D)	(None, 49, 49, 64)	0
conv2d_1 (Conv2D)	(None, 47, 47, 96)	55,392
max_pooling2d_1 (MaxPooling2D)	(None, 23, 23, 96)	0
conv2d_2 (Conv2D)	(None, 21, 21, 128)	110,720
max_pooling2d_2 (MaxPooling2D)	(None, 10, 10, 128)	0
conv2d_3 (Conv2D)	(None, 8, 8, 96)	110,688
max_pooling2d_3 (MaxPooling2D)	(None, 4, 4, 96)	0
conv2d_4 (Conv2D)	(None, 2, 2, 64)	55,360
max_pooling2d_4 (MaxPooling2D)	(None, 1, 1, 64)	0
flatten (Flatten)	(None, 64)	0
dense (Dense)	(None, 64)	4,160
dense_1 (Dense)	(None, 6)	390

Fig. 5: Architecture of CNN 1

We test this architecture with input dimensions of the images of:

- 100x100x3 pixels (Model 1)
- 128x96x3 pixels (Quarter resolution, Model 2)
- 512x384x3 pixels (Full resolution, Model 3)
- 256x192x3 pixels (Half resolution, Model 4)

And obtained the following loss and accuracy curves shown on Fig. 6. Model 1 was the one that overfitted the least, but the accuracy of the model wasn't as high as others. Model 2 was the one that overfitted the most. Models 3 and 4 are slightly overfitted. Model 3 was the best one with a validation F1-score of 0.87 and accuracy of 0.87.

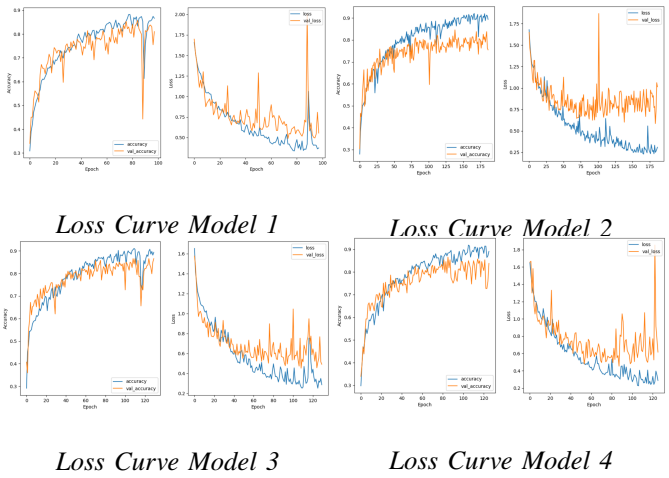


Fig. 6: Training Loss Curves for CNN Models with CNN 1 architecture

Through the analysis of the confusion matrix of model 3, shown in Fig. 7, we can see that the model tended to classify other classes as paper.

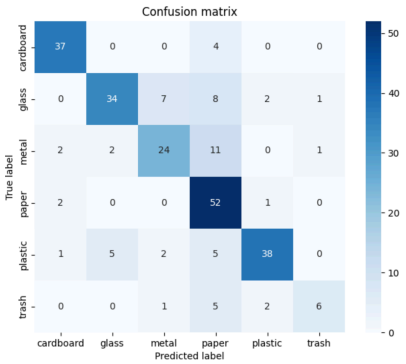


Fig. 7: Confusion matrix for Model 3

b) *CNN 2:*

Given the previous results, we built a new architecture with 6 blocks of *BatchNormalization*, *Conv2D*, *Dropout*, and *MaxPooling2D* layers, then a *Dense* layer with *ReLU* activation and L2 regularization, and a final *Dense* layer with softmax activation. The goal of this architecture was to improve the maximum accuracy by adding more *Conv2D* layers and by reducing overfitting by introducing *Dropout* layers and L2 regularization. This architecture is shown on Fig. 8.

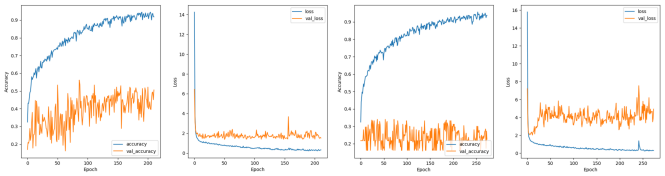
Layer (type)	Output Shape	Param #
batch_normalization_4 (BatchNormalization)	(None, 512, 384, 3)	12
conv2d_20 (Conv2D)	(None, 510, 382, 64)	1,792
dropout (Dropout)	(None, 510, 382, 64)	0
max_pooling2d_20 (MaxPooling2D)	(None, 255, 191, 64)	0
batch_normalization_5 (BatchNormalization)	(None, 255, 191, 64)	256
conv2d_21 (Conv2D)	(None, 253, 189, 64)	36,928
dropout_1 (Dropout)	(None, 253, 189, 64)	0
max_pooling2d_21 (MaxPooling2D)	(None, 126, 94, 64)	0
batch_normalization_6 (BatchNormalization)	(None, 126, 94, 64)	256
conv2d_22 (Conv2D)	(None, 124, 92, 128)	73,856
dropout_2 (Dropout)	(None, 124, 92, 128)	0
max_pooling2d_22 (MaxPooling2D)	(None, 62, 46, 128)	0
batch_normalization_7 (BatchNormalization)	(None, 62, 46, 128)	512
conv2d_23 (Conv2D)	(None, 60, 44, 128)	147,584
dropout_3 (Dropout)	(None, 60, 44, 128)	0
max_pooling2d_23 (MaxPooling2D)	(None, 30, 22, 128)	0
batch_normalization_8 (BatchNormalization)	(None, 30, 22, 128)	512
conv2d_24 (Conv2D)	(None, 28, 20, 256)	295,168
dropout_4 (Dropout)	(None, 28, 20, 256)	0
max_pooling2d_24 (MaxPooling2D)	(None, 14, 10, 256)	0
batch_normalization_9 (BatchNormalization)	(None, 14, 10, 256)	1,024
conv2d_25 (Conv2D)	(None, 12, 8, 256)	590,080
dropout_5 (Dropout)	(None, 12, 8, 256)	0
max_pooling2d_25 (MaxPooling2D)	(None, 6, 4, 256)	0
flatten_4 (Flatten)	(None, 6144)	0
dense_8 (Dense)	(None, 128)	786,560
dense_9 (Dense)	(None, 6)	774

Fig. 8: Architecture of CNN 2

We trained our new architecture with input size 512x384x3 pixels, and we varied the dropout parameter:

- 0.3 (Model 5)
- 0.5 (Model 6)

And obtained the following loss and accuracy curves shown on Fig. 9. Both of these models massively overfitted, but most surprisingly, increasing the dropout parameter caused the model to overfit more. The train accuracy is slightly better than the previous architecture.



Model 5 *Model 6*

Fig. 9: Training Loss Curves for CNN Models with CNN 2 architecture

Through the analysis of the confusion matrix of model 5, shown in Fig. 10, we can see that the model tended to classify other classes as paper, plastic, and cardboard, the most common classes.

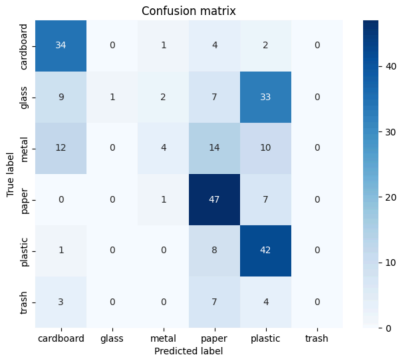


Fig. 10: Confusion matrix for Model 5

c) *CNN 3:*

To address class imbalance in the classification task observed in the above architecture, we computed class-specific weights based on the inverse frequency of each class in the training dataset. Specifically, if a class c contains n_c images and the total number of training images is N, its weight is calculated as $w_c = \frac{N}{n_c}$. These weights are then incorporated into a custom categorical cross-entropy loss function that penalizes errors in underrepresented classes more heavily. This weighted loss should ensure that the model learns effectively across all classes.

Noticing also that the *Dropout* layers seemed to further increase the overfitting problem, we decided to remove them and also increase the number of filters to obtain the architecture shown on Fig. 11.

Layer (type)	Output Shape	Param #
batch_normalization_4 (BatchNormalization)	(None, 512, 384, 3)	12
conv2d_20 (Conv2D)	(None, 510, 382, 64)	1,792
max_pooling2d_20 (MaxPooling2D)	(None, 255, 191, 64)	0
conv2d_21 (Conv2D)	(None, 253, 189, 96)	55,392
max_pooling2d_21 (MaxPooling2D)	(None, 126, 94, 96)	0
conv2d_22 (Conv2D)	(None, 124, 92, 128)	110,720
max_pooling2d_22 (MaxPooling2D)	(None, 62, 46, 128)	0
conv2d_23 (Conv2D)	(None, 60, 44, 128)	147,584
max_pooling2d_23 (MaxPooling2D)	(None, 30, 22, 128)	0
conv2d_24 (Conv2D)	(None, 28, 20, 128)	147,584
max_pooling2d_24 (MaxPooling2D)	(None, 14, 10, 128)	0
flatten_4 (Flatten)	(None, 17920)	0
dense_12 (Dense)	(None, 128)	2,293,888
dense_13 (Dense)	(None, 64)	8,256
dense_14 (Dense)	(None, 6)	390

Fig. 11: Architecture of CNN 3

The loss and accuracy curve shown in Fig. 12 shows that the overfitting is much less, but the accuracy is similar to models with the architecture CNN 1.

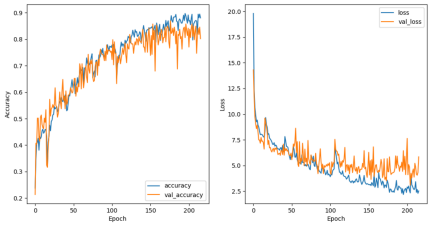


Fig. 12: Training Loss Curves for CNN Models with CNN 3 architecture

Through the analysis of the confusion matrix of model 7, shown in Fig. 13, we can see that the model tended to classify things more or less evenly, but the class trash (the least common) was wrongly predicted over other classes a few times.

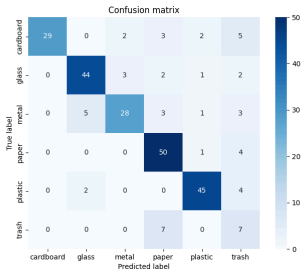


Fig. 13: Confusion matrix for Model 7

B. *Existing Architectures*

We first tested using a *ResNet50* architecture pre-trained on the ImageNet dataset. To do this, the convolutional layers of *ResNet50* were frozen, and only a new classification head composed of dense layers with *ReLU* and softmax activation was trained. Fine-tuning was then applied by unfreezing the last layers of the convolutional base. *ResNet50* introduces residual connections, consequently allowing intense network training without suffering gradient degradation, which should boost the performance. We obtained the results shown on Fig. 14, which show that the model underfitted.

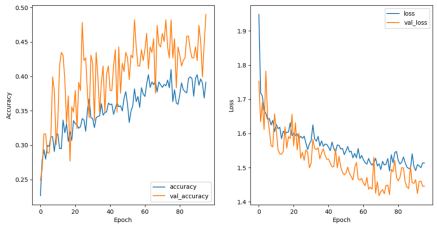


Fig. 14: Loss and accuracy curve for pretrained *ResNet50*

Since the previous approach underfitted, we decided to train the *ResNet50* from scratch and obtained the results shown on Fig. 15. The results show that the train accuracy is nearly 1, but the model overfitted. To note that we also tried to increase the number of layers that underwent fine-tuning in the above experiment, but the results were similar so we decided to just train from scratch.

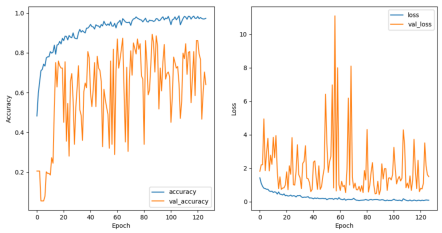


Fig. 15: Loss and accuracy curve for *ResNet50* trained from scratch

We wanted to test whether the overfitting was specific to the *ResNet50* architecture or if other architectures worked well in this domain so we tested 3 more architectures: *Xception*, *MobileNet*, and *EfficientNetB0*. The *Xception* architecture uses spatial and depth separation in the convolutions. The *MobileNet* architecture uses depth-separable convolutions to reduce complexity and optimize mobile devices. The *EfficientNetB0* architecture uses the compound scaling technique, a technique used to increase the width, depth, and resolution of images in a balanced way. We obtained the results shown on Fig. 16. Both the *EfficientNetB0* and *MobileNetV2* massively overfitted, but the *Xception* model showed the best results with an F1-score of 0.92 and accuracy of 0.92 for the test set.

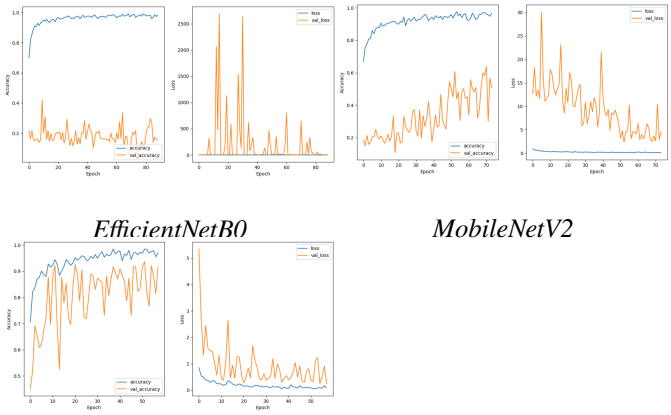


Fig. 16: Loss and accuracy curve for *EfficientNetB0*, *MobileNetV2* and *Xception* models

Through the analysis of the confusion matrix of the *Xception* model, shown in Fig. 17, we can see that the model tended to classify things more or less evenly.

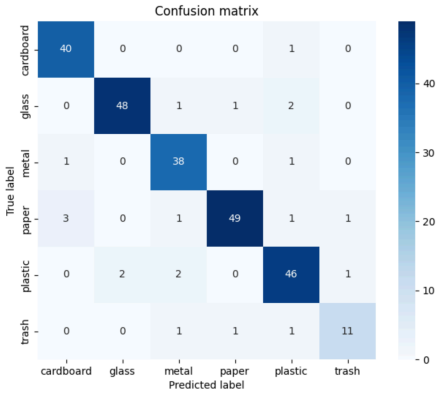


Fig. 17: Confusion matrix for *Xception* model

We also tried to add *Dropout* layers and L2 regularization in the classification head, and change the loss function. But the results were either worse or similar to the ones obtained above.

V. Results

A. CNN

The following table Table I shows the results of accuracy and F1-score that we obtain with each model for the Train and Test sets. Based on these results, Model 3 has the best performance.

TABLE I: Comparison between CNN models

Model	Accuracy (Test)	F1-score (Test)	Accuracy (Train)	F1-score (Train)
Model 1	0.81	0.81	0.90	0.90
Model 2	0.75	0.75	0.86	0.86
Model 3	0.87	0.87	0.91	0.91
Model 4	0.84	0.84	0.93	0.93
Model 5	0.51	0.40	0.96	0.96
Model 6	0.23	0.09	0.94	0.94
Model 7	0.80	0.81	0.88	0.88

B. Existing architectures

The following table Table II shows the results of accuracy and F1-score that we obtain with each model for the Train and Test sets. Based on these results, Model *Xception* has the best performance.

TABLE II: Comparison between CNN models

Model	Accuracy (Test)	F1-score (Test)	Accuracy (Train)	F1-score (Train)
ResNet50 (pretrained)	0.49	0.48	0.42	0.41
ResNet50 (trained)	0.64	0.63	0.98	0.98
EfficientNetB0	0.16	0.04	0.98	0.98
MobileNetV2	0.51	0.49	0.97	0.97
Xception	0.92	0.92	0.96	0.96

C. Comparison with state of the art

The following graph Fig. 18 shows the results of accuracy on Test sets across the state-of-the-art models and ours. Based on these results, we can see that [5] is by far the best model, but our results with the *Xception* model fall in line with results from the state-of-the-art.

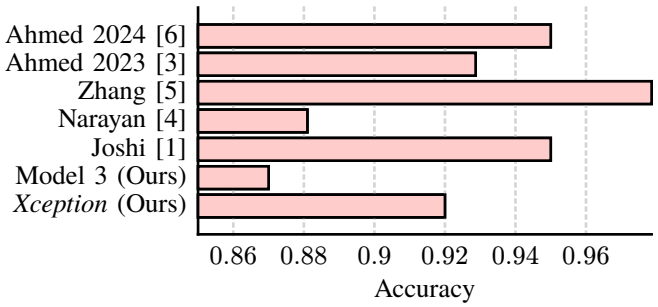


Fig. 18: Barchart of accuracy per model, State of the Art and Ours

VI. Results with Vision Transformer (ViT)

A Vision Transformer model was also tested for this work. In this model, ViT divides the image into small patches and applies attention to each patch in a similar way to the sequential processing of tokens in text.

However, the results obtained when using this model were significantly lower than those of CNNs in terms of accuracy and F1-score. However, considering that this type of model requires a large volume of data to be effective, the small size of the dataset may be one reason for these results. In addition, ViT's sensitivity to visual noise and spatial variations may also be reasons for this.

VII. Discussion

The analysis of the results allows us to draw several conclusions about the impact of the applied techniques.

Regarding the CNN developed from scratch, it was found that strategies like *Dropout* layers and L2 regularization didn't work well in this problem, likely due to the use of data augmentation that also helps with overfitting problems.

Furthermore, it should be noted that the high visual variability within the same class could also be one of the reasons for overfitting episodes, as this increases the complexity of the problem. Since this requires the model to learn more abstract and generalizable patterns.

The pre-trained feature models used in the transfer learning model didn't yield great results, but training them from scratch yields good results, depending on the architecture.

Furthermore, analyzing the techniques used and their contribution to improving performance revealed a notable difference in contribution. Thus, while the introduction of Dropout in Model 5 did not result in substantial improvements compared to Model 3, the adoption of the weighted loss function in Model 7 already led to a better confusion matrix.

That said, the *Xception* model has an advantage over the CNN model 3 in terms of accuracy. However, custom CNN remains a valid option, particularly in scenarios with computational resource constraints or when one wants complete control over the architecture. But, in real contexts, these factors become determinants. In this sense, *Xception* presented better metrics results; however, since it is a complex model with a high computational weight, its implementation in devices may be complex. As for the CNN model, in addition to maintaining a competitive performance, it is developed from scratch, making it a faster and lighter alternative.

Finally, comparing the results obtained with those found when developing the project, it can be seen that the values obtained with the *Xception* model (92% accuracy) were close to the values obtained in studies such as Narayan (2021) (88.1% accuracy) and Ahmed et al. (2023) (92.87% accuracy using InceptionV3).

The value of our approach is derived from the meticulous evaluation of custom-built CNNs against a range of established pre-trained architectures. This empirical investigation provides valuable insights into the practical trade-offs and performance nuances of these diverse approaches, thereby offering a useful reference for future work in automated waste classification, particularly in scenarios with similar dataset characteristics.

VIII. Conclusions

This study sought to explore the use of deep learning techniques in automatically classifying household waste. Using the "Garbage Classification" dataset available on Kaggle, two main approaches were tested for the six waste categories: custom convolutional networks developed from scratch and training existing architectures.

The results demonstrate that the best performance was obtained by the *Xception* model. However, the custom CNN model also achieved quite competitive results, which allows us to conclude that in situations with a restriction on computational resources or a need for more direct control over the network architecture, models developed from scratch remain valid and robust.

Furthermore, some difficulties were encountered during the work's development, namely, overfitting. This is likely due to the diversity present in each class, coupled with the relatively low number of images, which prevented the models not learn more general features.

It would be interesting to carry out a more in-depth study of this problem through explanatory methods, such as Grad-CAM. And study more effective techniques to reduce overfitting and improve the quality of results by using k-fold cross-validation, which wasn't used because of the long training times.

References

[1] I. Joshi, P. Dev, and G. Geetha, "Garbage Classification Using Deep Learning," in *2023 International Conference on Circuit Power and Computing Technologies (ICCPCT)*, Kollam, India, 2023, pp. 809–814. doi: 10.1109/ICCPCT58313.2023.10245133.

[2] Eurostat, "Waste statistics." [Online]. Available: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Waste_statistics

[3] M. I. B. Ahmed *et al.*, "Deep Learning Approach to Recyclable Products Classification: Towards Sustainable Waste Management," *Sustainability*, vol. 15, no. 14, p. 11138, 2023, doi: 10.3390/su151411138.

[4] Y. Narayan, "DeepWaste: Applying Deep Learning to Waste Classification for a Sustainable Planet." [Online]. Available: https://arxiv.org/abs/2101.05960

[5] X. Zhang, L. Guo, J. Xie, and S. Wang, "Fine-Tuning Models Comparisons on Garbage Classification for Recyclability." [Online]. Available: https://arxiv.org/abs/1908.04393

[6] A. Ahmed, M. Said, and M. Chowdhury, "Advancing Recycling Efficiency: A Comparative Analysis of Deep Learning Models in Waste Classification." [Online]. Available: https://arxiv.org/abs/2411.02779

[7] asdasdasasdas, "Garbage classification." [Online]. Available: https://www.kaggle.com/datasets/asdasdasasdas/garbage-classification