**GARBAGE CLASSIFICATION**

Angela Lozanovska & Davor Stefanovski

Faculty of Electrical Engineering and Information Technologies

University of Ss. Cyril and Methodius in Skopje, Macedonia

[lozanovskaangela@gmail.com](mailto:lozanovskaangela@gmail.com) [davorflute@gmail.com](mailto:davorflute@gmail.com)

**Abstract**

This project explores the application of transfer learning and convolutional neural networks (CNNs) for image recognition. Accurate classification of waste is a critical step towards improving recycling efficiency, reducing environmental pollution, and promoting sustainable waste management practices.

The methodology includes data preprocessing, augmentation, and feature extraction using pre-trained models such as MobileNetV2 and VGG16. On the other hand, CNNs are designed for tasks involving images and here the work is done by applying convolutional filters to extract different features.

The models are evaluated using accuracy to ensure robust performance. The results demonstrate that the implemented approach achieves high classification accuracy, showcasing the effectiveness of combining transfer learning and CNNs for garbage classification tasks. This study underscores the potential of deep learning models in advancing automated waste segregation systems.

**Introduction**

The growing global waste problem has made effective waste management a critical priority for governments, organizations, and communities. Accurate classification of garbage plays a vital role in improving recycling processes, minimizing landfill usage, and reducing environmental pollution. However, manual sorting of waste is time-consuming, labor-intensive, and prone to errors, making automated garbage classification systems increasingly important.

With advancements in deep learning, technologies like Convolutional Neural Networks (CNNs) and transfer learning from other CNN networks have shown remarkable potential in image recognition tasks, including waste classification. The approach by transfer learning not only improves classification accuracy but also reduces the time and resources required to train models from scratch.

For this project, we used dataset from Kaggle:  
<https://www.kaggle.com/datasets/asdasdasasdas/garbage-classification>The dataset comprises approximately 2,527 images categorized into six distinct classes. To enhance model performance and address potential class imbalances, we employed data augmentation techniques, which expanded the dataset to 10,108 images. Following augmentation, the images were normalized to ensure consistent scaling of pixel values, and the data was subsequently split into training, validation, and testing subsets. This preprocessing step laid the foundation for building the models effectively.

**Materials and Methods**

The Garbage classification dataset is from Kaggle and it consists of 2527 images categorized into six distinct classes: cardboard, glass, metal, paper, plastic, trash. (Figure 1)

First, the dataset was loaded from Kaggle, and the images were reviewed to understand their characteristics and structure. Given the relatively small size of the dataset, data augmentation was applied to increase the number of images and improve model performance. The dataset was augmented using techniques such as rotation (randomly rotate images in the range 0-20 degrees), width shift (randomly shift images horizontally), height shift (randomly shift images vertically), shear (randomly shear images), zoom (randomly zoom into images) horizontal flip (randomly flip images horizontally), fill mode (fill in newly created pixels after rotation or shift). Due to limited resources, we applied data augmentation techniques, ultimately expanding the dataset to 10,108 images. This significantly enhanced the dataset's diversity and size, providing a robust foundation for training the model and improving its performance. (Figure 2)

Beside data augmentation, data preprocessing involved resizing the images to a uniform size, normalizing pixel values and splitting the dataset into training (80%) and testing (20%) subsets to ensure robust model evaluation. The original dataset was divided into folders named after their corresponding classes. By mapping these names to our list of class names, we generated a NumPy array of labels using their indexes.

For model development, transfer learning was implemented using pre-trained models such as MobileNetV2 and VGG16, which convolutional layers were frozen, leveraging the pre-trained models' learned representations to extract relevant features from the augmented dataset. The models were trained using the Adam optimizer and sparse categorical cross-entropy loss function, with early stopping applied to prevent overfitting, while still giving a chance to the model to train more epochs.

Model performance was evaluated primarily based on accuracy, assessed both on the entire dataset and individually for each of the six classes. This detailed evaluation ensured a comprehensive understanding of the model's effectiveness in classifying various types of waste. All experiments were conducted using Python with TensorFlow, Keras, and NumPy on a T4 GPU provided by Google Collab, which significantly reduced training time compared to running the models on our device’s CPUs. The NVIDIA T4 GPU, optimized for deep learning tasks, offers high performance with its Tensor Cores, making it ideal for computationally intensive tasks like training Convolutional Neural Networks (CNNs).

**Results**

Although the dataset was relatively small - it was of good quality. To show the model that the data can be diverse within the same class and its accuracy, we applied data augmentation, generating three augmented images per original image. This process expanded the dataset significantly while still within our hardware resource boundaries. The data was split into training and testing sets in an 80:20 ratio. To standardize the data, all pixel values were normalized by dividing by 255, ensuring that the values ranged from 0 to 1.

* First model

The first model implemented for this project was with transfer learning techniques using MobileNetV2, a lightweight convolutional neural network developed by Google. MobileNetV2 is pre-trained on the ImageNet dataset, which contains over 14 million images across 1,000 categories. This pre-trained model is known for its efficiency in extracting features, making it ideal for transfer learning tasks like ours. Due to limited computational resources, the images were resized to *128×128* to reduce memory and processing demands. The pre-trained MobileNetV2 was configured to exclude its top classification layers, allowing customization for the six-class garbage classification task.

The base model of MobileNetV2 was frozen, meaning its pre-trained weights were not updated during training. This approach allowed us to leverage the features already learned from ImageNet while focusing computational resources on training the additional custom layers added for our task. These layers included a

1. Global Average Pooling layer
2. two fully connected Dense layers with ReLU activation (containing 128 and 64 neurons, respectively)
3. final Dense layer with a Softmax activation function for classifying the images into six categories. (Figure 3)

The model was compiled using the Adam optimizer with a learning rate of 0.0001. The loss function chosen was Sparse Categorical Crossentropy, appropriate for multi-class classification tasks with integer labels. (Figure 4) The training metric was accuracy. To prevent overfitting and optimize training efficiency, early stopping was implemented. (Figure 5) The early stopping mechanism monitored validation accuracy and stopped training if no improvement was observed for three consecutive epochs. Additionally, the model’s weights were restored to the best-performing ones based on validation accuracy, ensuring optimal performance.

The model was trained for up to 30 epochs, with a validation split of 20% of the training data. Despite the relatively small dataset and limited resources, the model achieved an overall test accuracy of 89%, demonstrating its effectiveness. Performance trends, including training and validation loss as well as training and validation accuracy, were plotted to evaluate the model's learning behavior over the epochs. (Figure 6)

To gain deeper insights, predictions were analyzed for each class. The best accuracy was observed for the "Cardboard" class, likely due to its consistent color and features. Other categories, such as "Glass," "Metal," "Paper," and "Plastic," showed satisfactory performance. As expected, the "Trash" category had the lowest accuracy, likely due to its diverse and irregular features, making it more challenging to classify. (Figure 7)

Overall, the results highlight the effectiveness of the MobileNetV2-based approach in garbage classification, with excellent performance in most categories and room for improvement in handling ambiguous classes like "Trash."

* Second model

The second model we used for this project also utilized transfer learning, this time with VGG16, a convolutional neural network architecture developed by the Visual Geometry Group at Oxford. VGG16 is well-known for demonstrating that using lots of parameters and layers can be useful in extracting deep features, as it consists of 16 layers (13 convolutional layers and 3 fully connected layers) and was trained on the extensive ImageNet dataset containing over 14 million images across 1,000 classes.

As with the first model, the input images were resized to *128×128* to reduce computational demands. The top layers of the VGG16 base model were excluded, allowing us to build a custom classifier for the six classes in our dataset. Upon examining the VGG16 base model, it had a total of 14,714,688 parameters, all of which were trainable, but these were frozen during training to retain the pre-trained feature extraction capabilities and focus the training on the custom classifier layers.

To build the custom model, we added layers on top of the frozen VGG16 base. First, a Global Average Pooling layer was included to reduce the spatial dimensions of the features while retaining the most important information. Next, densely connected neural network (NN) layers were added:

1. A fully connected layer with 512 hidden units using ReLU activation, followed by Batch Normalization for scaling activations and Dropout with a rate of 0.2 to prevent overfitting.
2. Another fully connected layer with 128 hidden units using ReLU activation, again followed by Batch Normalization and Dropout.
3. A final Dense layer with 6 units and a Softmax activation function was added for classifying the input into one of the six categories. (Figure 8)

The model was compiled with the Adam optimizer (learning rate = 0.0001), Sparse Categorical Crossentropy as the loss function, and accuracy as the evaluation metric. Early stopping, as defined earlier, was used to monitor validation accuracy and halt training if no improvement was observed for three consecutive epochs, restoring the best weights.

The VGG16-based model achieved a test accuracy of 84%, slightly lower than the MobileNetV2-based model. When evaluating the training and validation loss and accuracy over the epochs, the results were less stable compared to the first model, indicating more fluctuation during training. (Figure 9)

Class-specific predictions followed the same trend as the first model. The "Cardboard" class again achieved the highest accuracy due to its consistent features and simple structure, while the "Trash" class had the lowest accuracy, attributed to its variability and lack of defining characteristics. However, all class accuracies were slightly lower compared to the first model, reaffirming the superior performance of MobileNetV2 for this task. (Figure 10)

* Third model

For our third model, we decided to move away from transfer learning and build a convolutional neural network (CNN) from scratch, trained solely on our dataset. Convolutional Neural Networks are a class of deep learning models particularly well-suited for image classification tasks. They work by automatically and adaptively learning spatial hierarchies of features through convolutional layers, pooling layers, and activation functions. Unlike transfer learning, where we leverage pre-trained models, designing and training a CNN from scratch allows us to tailor the architecture to our specific dataset and task.

The architecture of our custom CNN model was relatively complex and included several convolutional layers grouped into blocks. Each block consisted of two convolutional layers, followed by Batch Normalization for stabilizing and accelerating training, a MaxPooling layer to reduce spatial dimensions, and Dropout to minimize overfitting. The detailed structure of the model included:

1. **First block:**
   1. Two Conv2D layers with 32 filters, kernel size *3×3*, and ReLU activation.
   2. Batch Normalization after each Conv2D layer.
   3. MaxPooling with a pool size of *2×2*.
   4. Dropout with a rate of 0.2.
2. **Second block:**
   1. Two Conv2D layers with 64 filters, kernel size *3×3*, and ReLU activation.
   2. Batch Normalization and MaxPooling as above, followed by Dropout.
3. **Third block:**
   1. Two Conv2D layers with 128 filters, kernel size *3×3*, and ReLU activation.
   2. Batch Normalization, MaxPooling, and Dropout applied as before.
4. **Fourth block:**
   1. Two Conv2D layers with 256 filters, kernel size *3×3*, and ReLU activation.
   2. Batch Normalization, MaxPooling, and Dropout applied similarly.
5. **Fully connected layers:**
   1. A Global Average Pooling layer to condense the feature maps.
   2. A dense layer with 512 units, ReLU activation, Batch Normalization, and a Dropout rate of 0.4.
   3. A final dense layer with 6 units and Softmax activation for multi-class classification. (Figure 11)

The model was compiled using the Adam optimizer with a learning rate of 0.0001, Sparse Categorical Crossentropy as the loss function, and accuracy as the evaluation metric. It was trained over 30 epochs with a batch size of 32 and a validation split of 20%. Unlike the previous models, we did not use early stopping for this model, as we wanted to observe its performance over the full range of epochs, even if validation loss became unstable. (Figure 12) This will be later discussed.

Despite being a custom model trained entirely on our limited dataset, it achieved a test accuracy of 86%, surpassing the results of the VGG16-based model and falling only slightly short of MobileNetV2. This result was unexpected and highlighted the capability of the custom CNN to learn from scratch effectively.

We evaluated the training and validation loss and accuracy, which showed noticeable instability in the validation metrics, reflecting the challenge of training a deep network on a small dataset. When analyzing predictions per class, the "Cardboard" category again achieved the highest accuracy, performing better than in the previous models. Conversely, the "Trash" category remained the weakest, with accuracy lower than the other models, due to its inherent variability and lack of clear defining features. (Figure 13)

**Discussions**

We decided to use MobileNetV2 and VGG16 as the two transfer learning models for our experiments. Both models are widely recognized for their effectiveness in image-related tasks, but they differ significantly in terms of architecture, purpose, and training datasets.

**MobileNetV2** is designed to be a lightweight and efficient model, making it particularly suitable for mobile and embedded devices. It utilizes depth wise separable convolutions to reduce the computational cost while maintaining accuracy. MobileNetV2 is pre-trained on the **ImageNet dataset**, which contains over 1.2 million labeled images across 1,000 classes. Its efficiency makes it a great choice for tasks where computational resources are limited or for applications that require faster inference times.

**VGG16**, on the other hand, is a deeper and more traditional convolutional neural network with 16 layers. It is known for its simple and consistent architecture, using only 3x3 convolutions stacked on top of each other, followed by fully connected layers. Like MobileNetV2, VGG16 is also pre-trained on the **ImageNet dataset**, but due to its larger size and computational complexity, it is less suited for resource-constrained environments. However, VGG16 often performs better on datasets with high visual complexity due to its depth and capacity to capture finer details.

When comparing the two, **MobileNetV2 typically gives better results on smaller datasets or when computational efficiency is crucial**, as it balances accuracy and speed. **VGG16, however, may yield slightly better accuracy on larger datasets or more complex images**, but at the cost of increased computational requirements and longer training times.

In terms of our experiments, the results depended not only on the models themselves but also on the dataset size and characteristics. Both models were trained on the reduced image size of 128x128 pixels, which impacted their performance. MobileNetV2 achieved slightly higher accuracy due to its optimized design for smaller inputs and resource-constrained scenarios, whereas VGG16, despite being a deeper model, was more prone to overfitting and performed slightly worse on our dataset.

In the context of convolutional neural networks (CNNs), datasets are typically categorized based on the number of labeled images available for training. While there is no strict definition, the following general ranges are commonly used:

* **Small Datasets**: These typically contain fewer than **50,000 images**. Small datasets often lead to challenges such as overfitting, as the model may not have enough diversity in the training data to generalize well to unseen data. Techniques like data augmentation, transfer learning, or using simpler models are often employed to improve performance on small datasets.
* **Medium Datasets**: These generally include between **50,000 and 1 million images**. Medium-sized datasets allow for training more complex models without excessive overfitting, but still may not be sufficient to fully train very deep architectures without pretraining.
* **Large Datasets**: These consist of over **1 million images** and are typically used to train large, complex CNN architectures from scratch. Examples include ImageNet (over 1.2 million labeled images) and Open Images Dataset (over 9 million images). Large datasets allow models to learn rich feature representations, leading to better generalization.

In our case, the dataset contained **10,000 images**, which is considered small. Given this, we expected the results to be modest due to the limited amount of training data. However, we were positively surprised by the performance of the models. This could be attributed to several factors, including the use of **data augmentation**, which artificially increased the size and variability of the dataset, and the application of **transfer learning**, which leveraged pretrained models like MobileNetV2 and VGG16 to extract meaningful features from the images. These techniques helped mitigate the limitations of the small dataset and yielded surprisingly good accuracy results.

As expected, the first model gave us better results – 89% accuracy, and the second model – 84% accuracy.

It would likely have been better if we had not reduced the image size from **250×250** to **128×128**, as the larger image size would have preserved more details and potentially improved the model's performance. However, due to resource constraints, we had to work with the reduced size. The google collab free tier virtual machine, simply didn’t have enough RAM to accommodate them. To compensate for this limitation, we invested considerable effort in optimizing the models, experimenting with various layers, neuron configurations, and activation functions. The models we finalized were the best we could achieve under these constraints, providing the highest accuracy.

As mentioned earlier, we compiled all models using the same settings: **Adam optimizer**, **sparse categorical cross-entropy** as the loss function, and **accuracy** as the evaluation metric. We also implemented **early stopping** in the first two models, which used transfer learning. Early stopping helped achieve the best results in these models by preventing overfitting and saving computational resources. We set **patience = 3**, allowing the training process to tolerate up to three consecutive decreases in accuracy before halting.

Initially, we planned to apply early stopping to our custom CNN model as well. However, the custom model was inherently unstable, with accuracy fluctuating significantly between epochs—sometimes achieving 70% in one epoch and dropping to 30% in the next. When we applied early stopping, there were instances where the model stopped training within the first few epochs, leaving us with a poor accuracy of around 40%. This was not reflective of the model's potential but rather a result of its instability causing multiple consecutive drops in accuracy.

Despite experimenting with various adjustments to early stopping, we could not achieve satisfactory results. Therefore, we decided to train the custom model for a fixed **30 epochs** without early stopping. While the model's accuracy continued to vary between epochs, the overall results were surprisingly good, with a final accuracy of **86%**. This demonstrated that, even without the aid of early stopping, the custom model was able to learn and generalize effectively over the dataset, showcasing its potential despite its instability.

The instability of CNN training can be caused by several factors, and **small datasets** can definitely be one of them. Here are potential causes for the unstable CNN training, along with the impact of a small dataset:

1. **Small Dataset:**
   1. **Limited Generalization:** A small dataset does not provide enough diverse examples for the model to learn from, which can lead to poor generalization and overfitting. This might cause large fluctuations in accuracy between epochs, as the model struggles to learn patterns that can generalize to new data.
   2. **Instability in Weight Updates:** With fewer data points, the model's weight updates can be less stable, leading to significant variations in the loss and accuracy from epoch to epoch. The model may learn certain features well in one epoch, but fail to do so in the next, especially if the data does not provide enough variety.
2. **Learning Rate:**
   1. A high learning rate can cause large fluctuations in the loss and accuracy during training, resulting in unstable training. Even with a small dataset, the optimizer might make overly large updates to the model's weights, preventing it from converging properly.
   2. On the other hand, too small a learning rate can result in slow convergence or getting stuck in local minima, although it tends to be less unstable than a high learning rate.
3. **Complexity of the Model:**
   1. Your custom CNN model might be too complex for the small dataset. Deep models with many layers and parameters can overfit small datasets, especially when there's insufficient data to properly train all layers. This can lead to instability, as the model tries to fit noise rather than meaningful patterns in the data.
4. **Insufficient Regularization:**
   1. If your model lacks sufficient regularization techniques (such as dropout or batch normalization), it might be prone to overfitting and instability. Regularization helps the model generalize better, reducing the variance in training and validation results.
   2. The large variance in accuracy during epochs in your custom model could be indicative of overfitting, where the model performs well on the training set but poorly on the validation set.
5. **Data Augmentation:**
   1. Data augmentation can help stabilize training by artificially increasing the size of the training dataset and introducing more variety in the images. However, if not applied correctly, or if the augmentation strategies are too aggressive, it can also cause instability, especially if the augmentation doesn't closely match the real-world data distribution.

The small dataset likely played a significant role in the instability of the custom CNN model. Small datasets tend to cause overfitting, instability in weight updates, and poor generalization, which is why transfer learning with models like **MobileNetV2** and **VGG16** performed better and were pretty stable. These models were pretrained on large, diverse datasets, allowing them to handle small datasets more effectively. By using pretrained models, we avoid training from scratch, which can be unstable and prone to fluctuations in performance.

One of our best ideas was to evaluate the performance of the models for each class. This allowed us to understand where the models excelled and where they struggled. Based on our observations, we predicted that the models would perform best in the **cardboard** class because the images in this category were relatively simple and lacked intricate details. On the other hand, we expected the models to perform worst in the **trash** class due to the diversity of items in this category, which made it harder for the models to identify consistent features.

Here are the results for each model across the six classes:

* **MobileNetV2 (Model 1):**

92.9%, 87.2%, 88.7%, 90.1%, 86.0%, 86.5%

* **VGG16 (Model 2):**

92.5%, 81.2%, 86.4%, 85.3%, 79.9%, 72.1%

* **Custom CNN (Model 3):**

96.1%, 83.9%, 89.5%, 84.3%, 78.8%, 75.9%

From these results, it is clear that the **first model using MobileNetV2** consistently delivered the best overall performance across the classes. The **custom CNN model** followed closely, achieving the highest accuracy for the cardboard class but showing more variation in performance for other categories. The **VGG16 model**, on the other hand, was the weakest, particularly in the trash class, where it performed poorly compared to the others.

Overall, MobileNetV2 proved to be the most effective model in this task, demonstrating its ability to generalize well across the dataset. The custom CNN performed admirably despite being trained from scratch, surpassing VGG16, which struggled with the complexity and diversity of some classes. This analysis confirmed the importance of evaluating models on a class-by-class basis to better understand their strengths and weaknesses. (Figure 14)

**Figures**

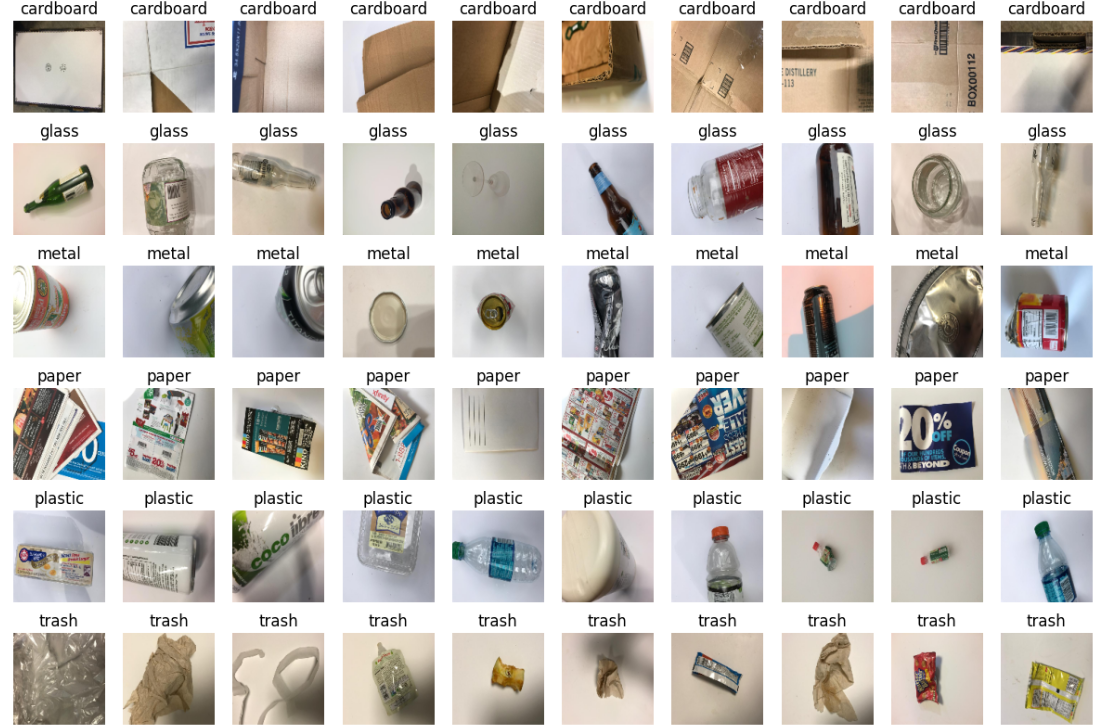


Figure 1



Figure 2



Figure 3

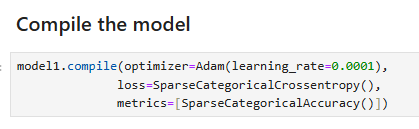


Figure 4

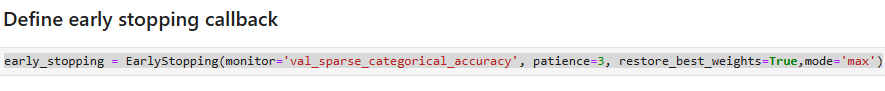


Figure 5

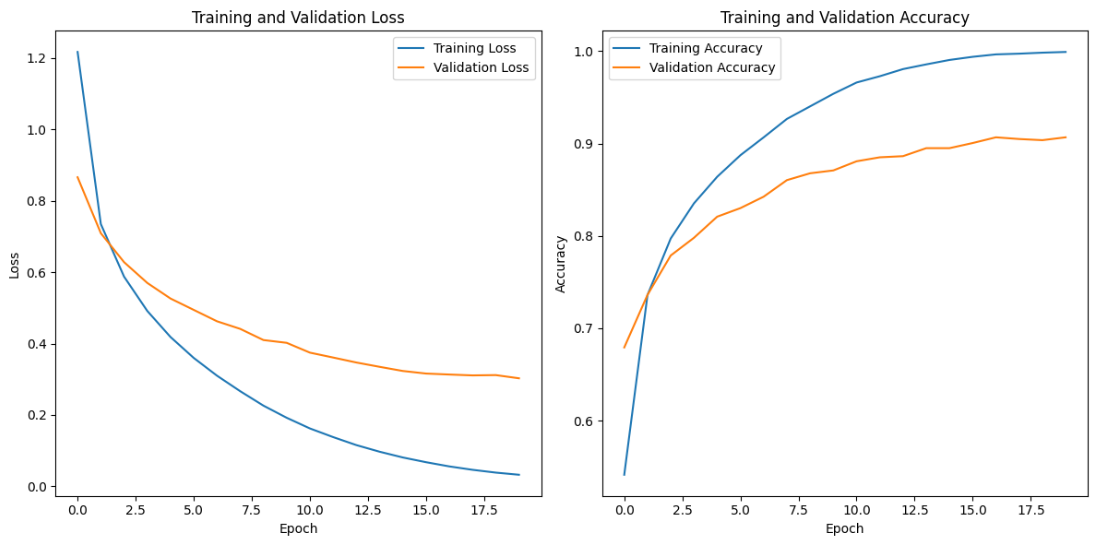


Figure 6

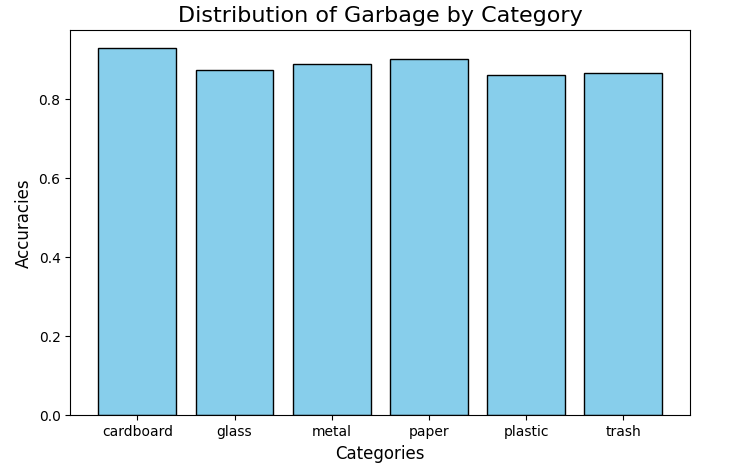


Figure 7

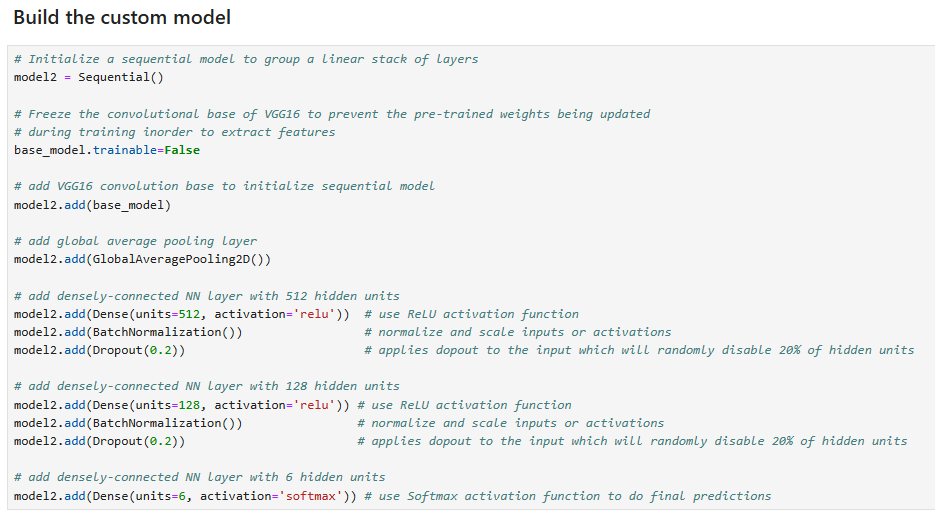
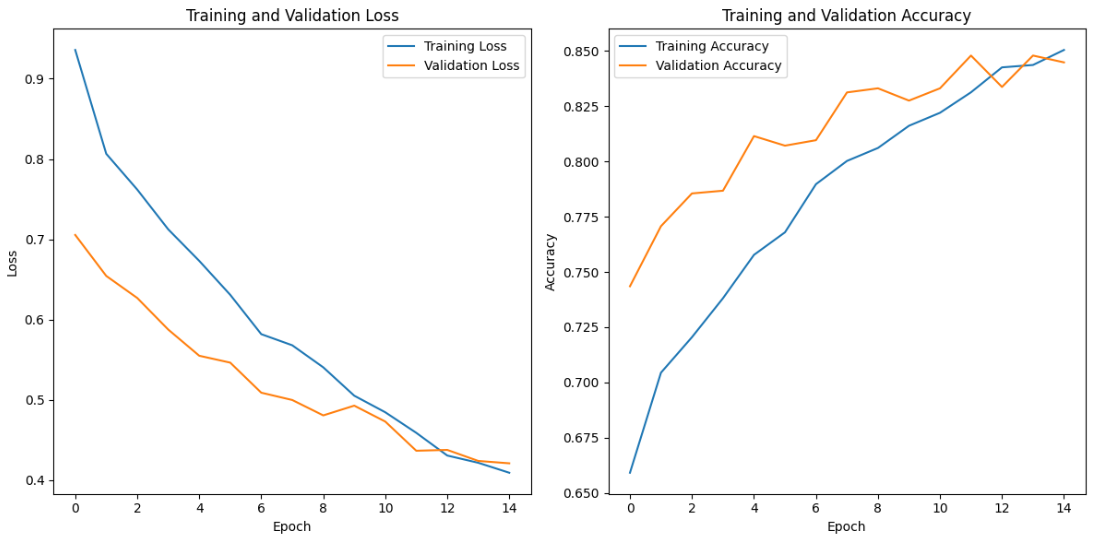


Figure 8

Figure 9

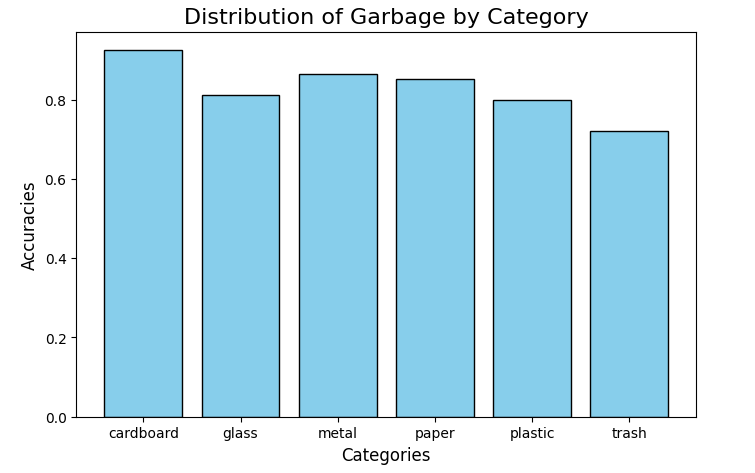
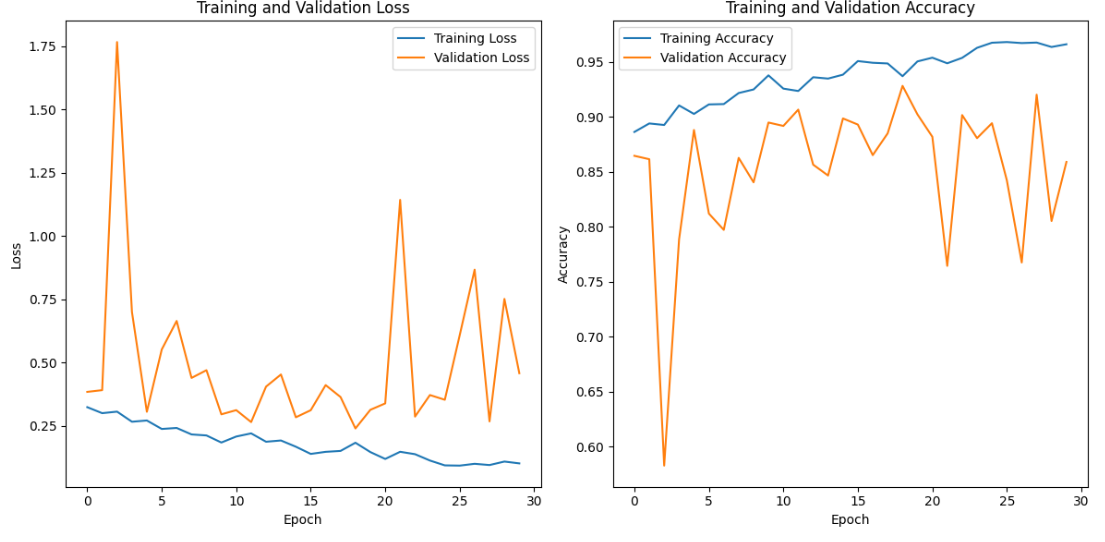


Figure 10



Figure 11

Figure 12

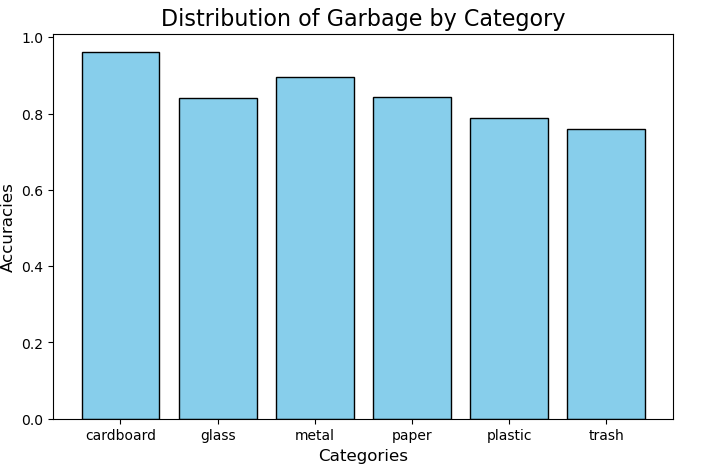


Figure 13

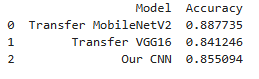


Figure 14

References:

1. Dataset from Kaggle

<https://www.kaggle.com/datasets/asdasdasasdas/garbage-classification>