# Real-time Garbage Classification using CNN Architecture

* *Gowtham (VCET - IIIrd Year)*

## Problem Statement :

The project aims to develop a real-time garbage classification system capable of identifying different types of waste materials through computer vision. This system will help automate waste sorting processes and improve recycling efficiency.

## Proposed Solution

The proposed solution is a custom Convolutional Neural Network (CNN) designed for waste classification. It processes 300x300x3 RGB images to classify them into six categories: cardboard, glass, metal, paper, plastic, and trash. The architecture includes four convolutional blocks with Conv2D layers (filters: 32→64), ReLU activation, and MaxPooling2D for size reduction. After feature extraction, a Flatten layer converts data into a 1D vector, followed by two Dense layers (64 and 32 neurons) with ReLU and Dropout (0.2) to prevent overfitting. The final Dense layer uses softmax activation for multi-class classification. The model is trained with categorical crossentropy loss and the Adam optimizer, ensuring efficient and accurate waste classification.

## Materials Required

### Hardware Requirements:

* Webcam for real-time detection
* Minimum 8GB RAM

### Software Requirements:

* Python 3.7+
* TensorFlow 2.x
* OpenCV
* Keras
* NumPy
* Matplotlib
* PIL (Python Imaging Library)

## Dataset Details

The project uses the TrashNet dataset, which contains 2527 images of garbage items across six categories:

* Dataset Source: [TrashNet on Kaggle](https://www.kaggle.com/datasets/feyzazkefe/trashnet)
* Image Resolution: 300x300 pixels
* Format: JPG
* Categories: 6 (cardboard, glass, metal, paper, plastic, trash)

## Code Repository

The complete project code & model is available in this google drive folder :

[CNN - Training & Inference](https://drive.google.com/drive/folders/17ziLAweoPCRQti_OE8c2olRcrDDicfK8?usp=drive_link)

Folder Structure :  
- Model File : [CNN\_trained\_model.keras](https://drive.google.com/open?id=1ro1bbAyhnPL-LtV_qGtG-DnMXa-uXMcD&usp=drive_copy)  
- Video Demo : [[DEMO] Realtime\_trash\_detection\_CNN.mp4](https://drive.google.com/open?id=1w1aUAnJpWY6-D_6Emhd6mpMNOfRJbANK&usp=drive_copy)  
- For running the model realtime : [realtime\_trash\_detection\_CNN.py](https://drive.google.com/open?id=1rTiHRG1OVXFT19U0_z2IuwgwmzZ7HE2b&usp=drive_copy)  
[- For training CNN using TrashNet : [CNN Training] - Jupyter Notebook - Trash Classification.ipynb](https://drive.google.com/open?id=1DI1pJmi21AdzwmhwUiZxb2CgGxEeq-uv&usp=drive_copy)  
- Results : [Test Images & Results](https://drive.google.com/open?id=1i_QkqSfr7VAhWskf9YnL2taHj3HMhEZf&usp=drive_copy)

## Implementation Procedure

#### 1. Preparing the Data

* **Dataset Organization:**  
  The dataset has images grouped in folders based on their type (e.g., plastic, glass, etc.).
* **Preprocessing:**
  + Images are resized to **300x300 pixels** to keep the input size consistent for the model.
  + Pixel values are scaled to a range of **0 to 1** (normalized). This makes calculations faster and avoids large values overwhelming the model.
* **Data Augmentation:**  
  Since having more diverse data helps models learn better, we use techniques like:
  + Flipping images horizontally and vertically.
  + Rotating, zooming, and shifting the images slightly.  
    This ensures the model can handle variations in new images.

#### 2. Designing the Model

A **CNN** is built step by step:

1. **Convolutional Layers:**  
   These layers scan the images for patterns like edges and textures. In the model:
   * The first layer detects 32 patterns using a 3x3 filter.
   * Later layers look for more complex features, using 64 filters of the same size.
   * Activation function: **ReLU** (Rectified Linear Unit) is used to introduce non-linearity, so the model can learn complex relationships.
2. **Pooling Layers:**  
   These reduce the size of the image data, focusing on key patterns and ignoring unnecessary details. The pooling size is **2x2**.
3. **Flattening Layer:**  
   This layer converts the reduced 3D data into a single list (1D) to prepare it for the final layers.
4. **Dense Layers:**  
   These are fully connected layers that learn to map the image patterns to specific categories.
   * Two hidden dense layers with **64 and 32 neurons** help the model learn effectively.
   * **Dropout** is applied to these layers to randomly ignore some neurons during training, which helps prevent overfitting.
5. **Output Layer:**
   * The final layer has **6 neurons**, one for each garbage category.
   * **Softmax activation** is used here, which ensures the output is a probability distribution.

#### 3. Training the Model

* **Loss Function:**  
  **Categorical Crossentropy** is used to calculate the error, as this is a multi-class classification problem.
* **Optimizer:**  
  **Adam optimizer** is chosen for efficient learning with adaptive adjustments.
* **Evaluation Metric:**  
  The model tracks **accuracy** during training and validation to monitor progress.

The model is trained for **50 epochs**, and after each epoch, it’s evaluated on both training and validation data.

#### 4. Saving the Model

The trained model is saved to a file (CNN\_trained\_model.keras) so it can be reused without retraining.

#### 5. Making Predictions

To classify a new image:

1. The image is resized to **300x300 pixels** and normalized (same as during training).
2. The model takes the processed image and predicts probabilities for each category.
3. The category with the highest probability is selected as the output.

For example, if a plastic image is input, the model might output something like:

* Plastic: 90%, Paper: 5%, Metal: 3%, etc.

Here, "Plastic" has the highest probability, so the image is classified as plastic.

### **How This Model Works Technically**

* The CNN learns by extracting **features** (like edges, textures, and shapes) from the input images in its convolutional layers.
* It combines these features to understand patterns unique to each garbage type.
* Through **training**, it adjusts its internal parameters to improve its predictions.

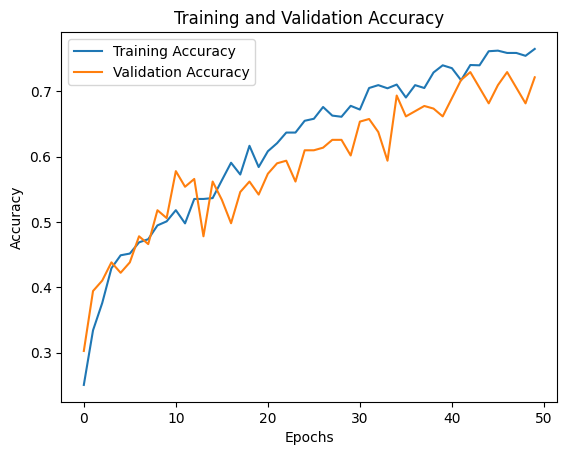
## Results

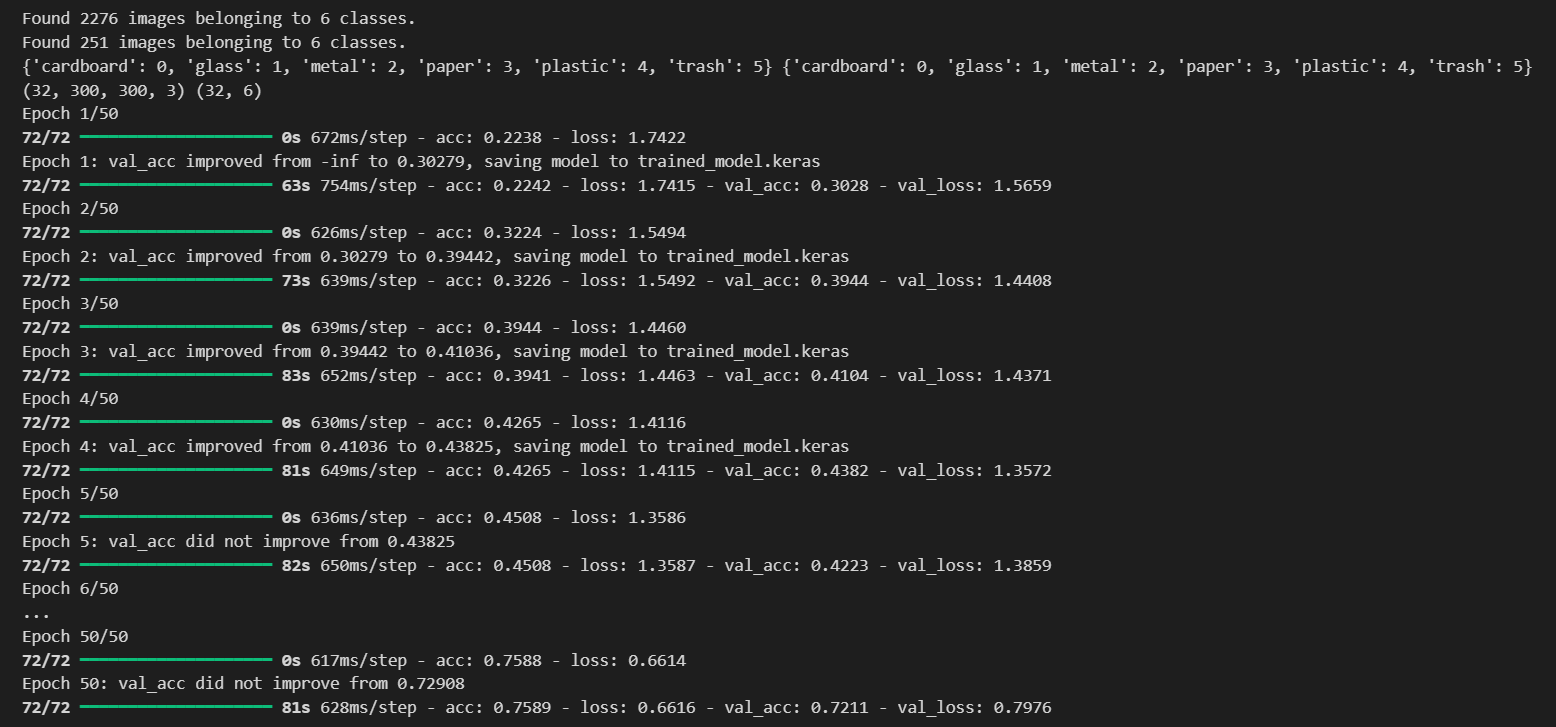
### Training Metrics

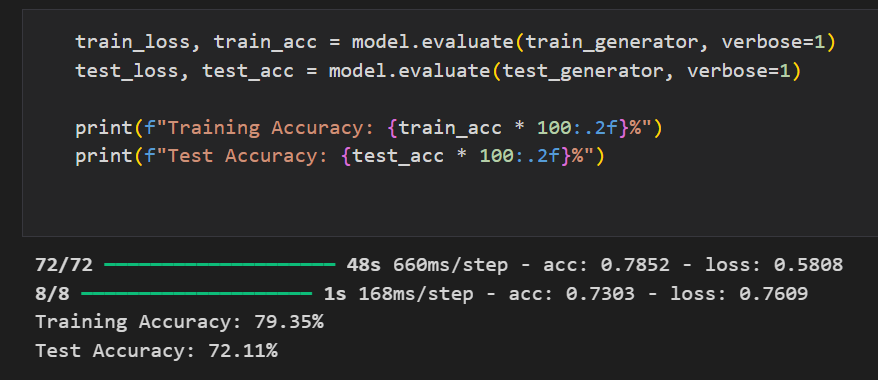
* Training Accuracy: 79.35%
* Validation Accuracy: 72.11%
* Model successfully exported in Keras format ([CNN\_trained\_model.keras](https://drive.google.com/open?id=1ro1bbAyhnPL-LtV_qGtG-DnMXa-uXMcD&usp=drive_copy))

### Real-world Performance

* Average inference time: ~40 ms per frame
* Reasonable accuracy in controlled lighting conditions
* Some limitations in varying light conditions







## Local Setup and Running

1. Download the given drive folder
2. Install dependencies:

pip install tensorflow opencv-python numpy pillow matplotlib

1. Download the trained model: [CNN\_trained\_model.keras](https://drive.google.com/open?id=1ro1bbAyhnPL-LtV_qGtG-DnMXa-uXMcD&usp=drive_copy)
2. Run real-time detection:

python [realtime\_trash\_detection\_CNN.py](https://drive.google.com/open?id=1rTiHRG1OVXFT19U0_z2IuwgwmzZ7HE2b&usp=drive_copy)