AI-Powered Waste Sorting System: Comprehensive Project Report

Dhruv Kannojia (590011908)
Dhruv.11908@stu.upes.ac.in

December 5, 2025

Abstract

This report details the implementation of a functional, integrated AI-Powered Waste Sorting System. The core achievement lies in successfully integrating a Deep Learning classification model (TensorFlow/Keras) with a relational Database Management System (MySQL). The primary objective of the DBMS is established as the analytical and auditing backbone, ensuring data persistence, enabling performance analysis, and providing the necessary feedback loop for AI model maintenance. The project methodology, detailed schema design, implementation of the Python server bridge for execution control, and analytical query results are presented herein.

1. Introduction and Objectives

1.1. Motivation and Problem Statement

The global challenge of municipal solid waste management is severely hindered by inefficient and inaccurate sorting processes. Manual sorting is costly, hazardous, and results in high contamination rates, significantly lowering the economic value of recycled materials. This project addresses this by proposing an autonomous solution. The primary goal is to shift from human-dependent sorting to a reliable, data-driven system.

1.2. Project Objectives

The central objectives of this project are twofold:

1. **ML Objective:** To develop and train a Convolutional Neural Network (CNN) capable of classifying waste images into 10 distinct categories with a verifiable accuracy ($\sim 88.7\%$).
2. **DBMS Objective (Core Focus):** To design and implement a robust MySQL database that functions as the central audit and analytical tool, capturing every AI decision in real-time, regardless of the sorting outcome.
3. **Integration Objective:** To create a seamless software pipeline

(Python) that connects image classification output to structured database logging, and remotely trigger this operation via a web interface.

1.3. Project Scope and Constraints

The project scope covers the design and implementation of the software pipeline from image input to final data logging and reporting. It includes the design of the CNN architecture, the creation of the MySQL schema, and the development of the Python integration scripts necessary for system control and execution. Constraint analysis was necessary due to the large dataset size and the requirement for real-time logging.

2. Literature Review

2.1. Context: AI in Waste Management

The application of Deep Learning in waste management is a rapidly growing field, primarily driven by advances in computer vision. Previous studies, such as those using the TrashNet dataset, demonstrate that transfer learning and customized CNN architectures (e.g., VGG16, ResNet) can achieve classification accuracy exceeding 90% under laboratory conditions. These findings provide the theoretical justification for using a CNN as the core classification model in this project.

2.2. Comparative Analysis of CNN Architectures

While more complex models like ResNet-50 offer superior accuracy due to skip connections, our resource constraints favored a custom shallow CNN. This balance ensures reasonable training time on consumer-grade hardware while retaining the core feature extraction capabilities needed for distinguishing material textures and shapes (e.g., distinguishing paper from cardboard).

2.3. Database Integration in IoT/ML Systems

In industrial settings (such as a Material Recovery Facility - MRF), raw data produced by sensors and AI systems must be handled by a transactional and scalable DBMS. The database provides ACID properties and ensures that critical operational data (such as bin capacity, material flow rate, and AI decisions) is logged reliably. Our selection of MySQL ensures compatibility with the chosen Python environment while providing the necessary relational integrity for complex reporting. The reliable logging of transient events is paramount in these high-throughput systems.

3. System Architecture and Methodology

3.1. Component Breakdown and Data Pipeline

The project employs a three-tier architecture model: Presentation (Web Interface), Application Logic (Python Scripts), and Data (MySQL/File System). The system's control and data flow demonstrates the fundamental interaction between the logical components:

- **Step 1: Input Trigger (Presentation):** The user selects images on the web dashboard, sending an API signal to the Python server.
- **Step 2: Control Layer (Python Bridge):** The Flask server receives the signal and executes the deploy_sorter.py script via subprocess.
- **Step 3: Classification (AI/ML):** The deploy_sorter.py script loads the CNN and processes images, generating the ai_prediction and prediction_score.
- **Step 4: Logging (DBMS):** The Python script immediately inserts the classification results into the MySQL SORTING_LOGS table.
- **Step 5: Physical Output:** Simultaneously, the image file is moved from UNSORTED_INPUT to the category-specific folder within OUTPUT_FOLDERS.

3.2. Methodology: Training Data Split

The total dataset size is $5,927$ images across 10 categories. The standard cross-validation methodology was used:

- **Training Set (70%): 4,149** images. Used by the CNN to learn features and adjust weights.
- **Validation Set (30%): 1,778** images. Used during training epochs to measure generalization and prevent overfitting.

4. DBMS Design and Implementation

4.1. Rationale for MySQL Selection

MySQL was chosen for its proven reliability, transactional integrity (crucial for logging every event once), and strong support for structured analytical queries (GROUP BY, COUNT, AVG), which are necessary for the project's reporting phase. The open-source nature ensured easy integration with the Python environment.

4.2. The SORTING_LOGS Schema Detail

The design focuses on capturing essential metadata and performance metrics.

- **Table: SORTING_LOGS**
  - log_id: INT (Primary Key, Auto Increment). Ensures unique identification of every classification event.
  - file_name: VARCHAR(255) (NOT NULL). The key link between the database log and the physical file system output.
  - ai_prediction: VARCHAR(50) (NOT NULL). Stores the result of the AI classification.
  - prediction_score: DECIMAL(5, 4). Crucially defined to ensure high precision for tracking confidence.
  - time_stamp: DATETIME (DEFAULT CURRENT_TIMESTAMP). Essential for real-time monitoring and chronological reporting.

4.3. Indexing Strategy

To ensure efficient execution of analytical queries on the SORTING_LOGS table, appropriate indexing was implemented. As the primary queries involve grouping by prediction and filtering by score/time, composite and single-column indices were defined:

- **Primary Index:** log_id (Clustered Index, automatically created).
- **Secondary Index 1 (Categorization):** Index on ai_prediction. Crucial for rapid execution of the COUNT and GROUP BY queries used in the Waste Flow Analysis.
- **Secondary Index 2 (Performance Audit):** Index on prediction_score. Optimizes the search for low-confidence records (WHERE prediction_score < 0.80)
-

5. AI/ML Model Implementation Detail

5.1. CNN Architecture and Training

The model is a shallow, custom CNN built using Keras's Sequential API. This architecture was chosen for its computational efficiency while retaining high classification accuracy.

- **L1-L2 (Convolutional Layers):** Two layers (Conv2D with 32 and 64 filters) extract hierarchical features (edges, textures).
- **Pooling Layers:** Two MaxPooling2D layers reduce the spatial dimensions, decreasing computation time and preventing overfitting.
- **Final Layers:** Flatten followed by two Dense layers, with the final softmax output layer classifying the features into 10 categories.

5.2. Hyperparameter Selection and Rationale

Hyperparameter tuning was conducted to optimize the model's performance on the small dataset:

- **Optimizer (Adam):** Selected for its adaptive learning rate properties.
- **Loss Function (categorical_crossentropy):** Required because the target labels are one-hot encoded (categorical classification).
- **Batch Size (32):** Provides a balance between training stability and memory usage on the development system.
- **Epochs (10):** Chosen empirically based on observing convergence in the validation accuracy.

5.3.Data Generator and Preprocessing

The ImageDataGenerator in Python handles image preprocessing, which is vital for feeding consistent data to the CNN:

- **Rescaling:** Normalizes pixel values from 0-255 to 0.0-1.0.
- **Resizing:** All images are resized to a fixed 128×128 pixel resolution.
- **Splitting:** Manages the 70% training and 30% validation split directly from the file system folders.

6. Integration and Control Implementation

6.1. The Python Control Script (deploy_sorter.py)

The deployment script is the operational heart of the system, combining the functions of the AI and the DBMS.

- **P1: Model Loading:** Loads the saved model (waste_sorter_model.h5).
- **P2: File I/O:** Uses the shutil library to move the file after successful logging, ensuring that an item is only moved once its database entry is confirmed.
- **P3: DBMS Connection Handling:** Manages the mysql.connector to open and close connections efficiently for each INSERT transaction, minimizing database overhead.

6.2. Web Server Bridge and Robustness

The Flask server (server_bridge.py) overcomes the browser's security limitations to execute local Python scripts.

- **Log Redirection:** The server executes the sorting script and redirects all console output into a temporary file (live_output.log) using subprocess.Popen. This enables the browser to stream logs.
- **Asynchronous Polling:** The web dashboard continuously polls the /api/get_logs endpoint to read the new lines from the log file, providing the live terminal experience.
- **CORS Security:** The implementation includes flask_cors to allow the browser (running on the local file system) to communicate securely with the server (running on port 5000), resolving cross-origin security issues.
-

7. Analytical Reporting and Results

7.1. Waste Frequency Analysis (Operational Intelligence)

This query groups results by prediction to provide immediate operational feedback on the volume distribution.

```
SELECT ai_prediction AS Category, COUNT(*) AS Total_Count FROM
SORTING_LOGS GROUP BY ai_prediction ORDER BY Total_Count
DESC;
```

7.2. Model Confidence Audit (Quality Control)

This query focuses on quality control, flagging records where the AI was uncertain. These low-confidence files require mandatory human review.

```
SELECT file_name, ai_prediction, prediction_score FROM
SORTING_LOGS WHERE prediction_score < 0.80 ORDER BY
```

prediction_score ASC;

8. Discussion, Challenges, and Future Scope

8.1. Demonstrating DBMS Value

The project successfully demonstrated the mandatory requirement: the DBMS is critical for system operation. By using SQL to filter for prediction_score < 0.80, we prove the database's role in proactive quality control. This transforms the AI's numerical output into an organizational task (human review queue).

8.2. Implementation Challenges

The primary implementation challenges were not related to the core CNN but to the integration layer:

- **Path Resolution:** Resolving relative path inconsistencies when executing Python scripts via the Flask server's subprocess was a major debugging hurdle (fixed by setting cwd=CODE_DIR).
- **Database Access:** Handling the Access denied (Error 1045) required ensuring consistent credential management between the configuration file and the MySQL user privileges.

8.3. Future Scope: Extending the DBMS Role

Future enhancements should focus on extending the DBMS's role to manage the physical infrastructure and maintain model metadata:

- **Automated Indexing:** Implementing database-side triggers to automatically update a separate RE_TRAIN_QUEUE table whenever a prediction_score falls below a threshold (0.75).
- **Bin Management (New Table):** Adding a WASTE_BINS table to track capacity and triggering alerts when a bin is 90% full.
- **Model Versioning (Metadata):** Integrating a MODEL_METADATA table to store the version, training date, and accuracy of the currently deployed model.

9. Conclusion

The AI-Powered Waste Sorting System achieved all stated objectives, showcasing the successful, real-world integration of machine learning and data management principles. The project confirms that the database serves as the essential **operational intelligence layer**, ensuring data integrity, performance measurement, and driving the necessary feedback loops for continuous system improvement. The project provides a scalable and robust framework for future development in smart waste solutions.