Cairo University

Faculty of Computers and Artificial Intelligence

Machine Learning Course

Project: Material Stream Identification System

Student Name	ID
Israa Mohamed Elsayed	20221252
Amany Mohamed Hussein	20221026
Malak Ahmed Eltabbakh	20221157
Rawan Amr Nabil	20221062
Mariam Eid Mohamed	20220330

TA: Eng. Mohamed Atta

1. Project Overview

The **Material Stream Identification (MSI)** System is an automated vision-based classification tool designed to identify and sort different types of post-consumer waste materials in real-time.

This system plays a vital role in enabling smart recycling facilities and advancing the goals of the circular economy — where waste is minimized, and valuable materials are recovered efficiently.

As part of this course project, you will build an end-to-end ML pipeline that:

- Takes raw images of waste items.
- Converts them into numerical feature vectors .
- Classifies them into one of 7 categories (6 material classes + 1 “Unknown”).
- And deploys the best-performing model into a real-time camera application.

We will implement and compare two foundational classifiers — SVM and k-NN — and justify your design choices throughout the process.

This project emphasizes mastery of the full ML lifecycle: from data preprocessing to deployment

— not just accuracy, but also robustness, interpretability, and real-world applicability.

Phase 1: Data Preparation & Augmentation

In this phase, we implemented a deterministic and reproducible data augmentation pipeline to meet the project's requirement of increasing the training sample size by at least 30% per class (Section 4.1). Our goal was not only to balance class distribution but also to enhance model robustness against real-world variations in lighting, orientation, and scale, while preserving the semantic integrity of waste materials.

1. Reproducibility Setup (Critical for Fair Evaluation)

To ensure consistent results across all team members and evaluation runs, we fixed all sources of randomness at the very beginning of the script:

- **os.environ['PYTHONHASHSEED'] = '0'**: Eliminates non-determinism in Python's hash-based operations (e.g., dictionary ordering).
- **np.random.seed(42) and random.seed(42)**: Fix the random number generators for NumPy (used by skimage) and Python's built-in random module.

This guarantees that every run produces the exact same augmented dataset, enabling fair comparison between models

2. Dataset Structure & Initial Validation

- The original dataset was organized into six folders corresponding to the classes:

  ["paper", "plastic", "metal", "glass", "cardboard", "trash"].

- We first copied all valid, non-corrupted images (those readable by cv2.imread) to a new directory (augmented_dataset), skipping any damaged files that could disrupt training.

3. Augmentation Strategy & Parameter Justification

We applied four augmentation techniques, carefully tuned to reflect realistic physical variations in waste items (not arbitrary distortions):

Technique	Parameter Range	Justification
Rotation	±15°	Waste items in real sorting systems may appear slightly tilted, but rarely exceed 15°. Larger angles (e.g., ±30°) would create unrealistic orientations that could mislead the model.
Horizontal Flip	50% probability (random.random() > 0.5)	Most waste items (bottles, cans, boxes) are symmetric horizontally. Flipping preserves material semantics while doubling pose variability.
Lighting Adjustment	Gamma ∈ [0.8, 1.2]	Simulates common illumination changes (e.g., shadows, bright reflections). Gamma < 1 brightens; gamma > 1 darkens. We avoided extreme values to prevent color distortion — critical for distinguishing materials like glass (transparent) vs. plastic (colored).
Scaling	Scale ∈ [0.95, 1.05] (±5%)	Mimics minor distance variations between the camera and object. Larger scales (e.g., ±10%) risk cropping out key features or introducing unrealistic object sizes.

4. Target Class Size & Balancing

We set a target of 500 images per class, which ensures:

- All classes exceed the minimum 30% increase (e.g., "trash" had only 110 originals → now 500 = +354%).
- Balanced training data, reducing classifier bias toward overrepresented classes like "paper" (originally 476 images).

5. Output Structure

The final augmented dataset is saved in:

data/augmented_dataset/

├── glass/

├── paper/

├── cardboard/

├── plastic/

├── metal/

└── trash/

Each subfolder contains original valid images + newly generated augmented images, ready for feature extraction.

Phase 2: Feature Extraction Methodology (Image → Vector)

To convert raw RGB images into fixed-length numerical feature vectors suitable for machine learning classifiers, we adopted a transfer learning-based approach using a pre-trained deep convolutional neural network (CNN). This methodology leverages high-level semantic representations learned from large-scale image datasets to encode material-specific visual cues.

2.1 Model Selection: MobileNetV2

We selected MobileNetV2 as our feature extractor due to its optimal balance of computational efficiency and representation quality. MobileNetV2 is a state-of-the-art CNN architecture designed for mobile and embedded vision applications. It utilizes depthwise separable convolutions and inverted residuals to achieve high accuracy with significantly reduced parameter count and computational cost compared to larger models like ResNet or VGG that make it more sutable for the real time application .

The model was loaded with weights pre-trained on the ImageNet dataset (1.28 million images across 1000 object classes). Critically, we froze all layers and excluded the final classification head (include_top=False), repurposing the network solely as a feature generator.

2.2 Feature Vector Construction

For each input image, the feature extraction pipeline is as follows:

Preprocessing: The raw image is loaded and converted from BGR (OpenCV format) to RGB. It is then resized to 224×224 pixels, the canonical input size for models pre-trained on ImageNet. The pixel values are normalized using the preprocess_input function, which applies the specific mean subtraction and scaling used during ImageNet training.

Forward Pass: The preprocessed image is passed through the MobileNetV2 network.

Global Average Pooling: The output from the final convolutional layer (a spatial feature map of size 7×7×1280) is reduced to a 1280-dimensional vector using Global Average Pooling (GAP) (pooling="avg"). This operation computes the mean activation across each of the 1280 feature channels, resulting in a compact, fixed-length descriptor that is invariant to small spatial translations.

The final output is a 1280-dimensional floating-point feature vector that captures high-level semantic content relevant to material identification, such as texture, shape, and object context.

Phase 3: Classifier Implementation & Training

Model A: k-Nearest Neighbors Classifier

The k-NN classifier was selected as one of the two foundational models for this project due to its simplicity, interpretability, and strong performance on small-to-medium datasets with well-defined feature spaces — characteristics that align with our handcrafted feature extraction approach.

Hyperparameter Selection via Grid Search

To ensure optimal performance, we performed an exhaustive hyperparameter search using GridSearchCV with 3-fold stratified cross-validation. We tuned three key parameters:

- n_neighbors: Tested values {3, 5, 7, 9, 11} to balance bias-variance trade-off.
- weights: Compared 'uniform' (all neighbors contribute equally) vs. 'distance' (closer neighbors have more influence).
- metric: Used 'euclidean' distance, which is mathematically appropriate for continuous, normalized feature vectors derived from CNN model.

The grid search identified the following optimal configuration:

{'n_neighbors': 3, 'weights': 'distance', 'metric': 'euclidean'}

Justification for Final Parameters

- **n_neighbors = 3**: A small k value was chosen because our dataset contains clear, separable material classes (e.g., metal vs. paper). A low k allows the model to capture local patterns without over-smoothing decision boundaries.

- **weights = 'distance'**: Distance-based weighting was preferred over uniform weighting because it assigns higher importance to closer neighbors — improving robustness against outliers and noisy samples.

- **metric = 'euclidean'**: Euclidean distance is the most natural choice for our dense, real-valued feature space, where geometric proximity reflects semantic similarity between materials.

Model Training and Evaluation

The final k-NN model was trained on the full training set (80% of data) after applying standardization. On the held-out validation set (20%), the model achieved **90.67% accuracy**. It demonstrated reasonable performance across most classes, particularly excelling on cardboard (F1-score: 0.86) and trash (F1-score: 0.90).

The model's simplicity and lack of training time make it suitable for rapid prototyping and edge deployment, though its sensitivity to high-dimensional noise and class imbalance limits its peak performance compared to SVM.

Model B: Support Vector Machine (SVM) Classifier

The Support Vector Machine (SVM) was selected as the second foundational classifier in this project due to its strong theoretical foundations, robustness to high-dimensional feature spaces, and proven effectiveness in non-linear classification tasks. Given the complex visual similarities between certain waste materials (e.g., plastic vs. glass), SVM is particularly well-suited to learning discriminative decision boundaries in the extracted feature space.

Hyperparameter Selection via Grid Search

To ensure optimal performance and avoid manual or arbitrary hyperparameter selection, we employed **Grid Search with cross-validation** to tune the SVM classifier's key hyperparameters. The search was conducted over a predefined parameter grid using stratified cross-validation on the training set, allowing us to evaluate each configuration fairly and select the one that maximized validation accuracy.

The grid search focused on the most influential SVM hyperparameters:

- **C (regularization parameter)**: controls the trade-off between margin width and classification error.
- **Gamma**: determines the influence of individual training samples when using the RBF kernel.
- **Kernel type**: evaluated to confirm the suitability of non-linear decision boundaries.

Based on the grid search results, the following configuration achieved the best validation performance and was selected for the final model:

- **Kernel**: RBF
- **C**: 50
- **Gamma**: scale
- **Probability estimation**: Enabled (probability=True)

Justification for Final Parameters

- **C = 50**: A relatively high regularization parameter was chosen to reduce classification errors on the training data while still maintaining generalization. This value provided a good balance between margin maximization and misclassification tolerance.
- **Gamma = scale**: Using the scale option allows gamma to adapt dynamically based on feature variance, preventing overfitting and eliminating the need for manual tuning.
- **Probability=True**: Enabling probability estimation allows the model to output calibrated class probabilities via Platt scaling, which is essential for implementing the "Unknown" rejection mechanism required in Phase 4.

Training and Preprocessing Pipeline

Prior to training, all feature vectors were standardized using **StandardScaler** to ensure zero mean and unit variance. This step is critical for SVM performance, as the algorithm is highly sensitive to feature scale.

The dataset was split into **80% training** and **20% validation** subsets using stratified sampling to preserve class distribution. The SVM model was trained on the standardized training data and evaluated on the held-out validation set.

Model Evaluation and Performance

On the validation set, the SVM classifier achieved a **validation accuracy of 93.83%**, outperforming the k-NN baseline. The model demonstrated particularly strong performance in distinguishing visually similar material categories, such as plastic and glass, due to its ability to learn complex non-linear decision boundaries.

Performance Evaluation between (KNN&SVM model):

KNN model: achieves 90.67% accuracy

✅ k-NN Validation Accuracy: 0.9067

	precision	recall	f1-score	support
glass	0.86	0.89	0.87	100
paper	0.95	0.86	0.90	100
cardboard	0.96	0.95	0.95	100
plastic	0.85	0.87	0.86	100
metal	0.90	0.89	0.89	100
trash	0.93	0.98	0.96	100
accuracy			0.91	600
macro avg	0.91	0.91	0.91	600
weighted avg	0.91	0.91	0.91	600

SVM model: achieves 93.83% accuracy

✅ SVM Validation Accuracy: 0.9383

	precision	recall	f1-score	support
glass	0.95	0.87	0.91	100
paper	0.97	0.91	0.94	100
cardboard	0.96	0.96	0.96	100
plastic	0.95	0.93	0.94	100
metal	0.87	0.97	0.92	100
trash	0.94	0.99	0.97	100
accuracy			0.94	600
macro avg	0.94	0.94	0.94	600
weighted avg	0.94	0.94	0.94	600

Both classifiers were trained on the same preprocessed dataset using identical train/test splits to ensure a fair comparison. The evaluation focused on validation accuracy, per-class F1-scores, computational efficiency, and suitability for real-time deployment.

Metric	k-NN (Best Config)	SVM (RBF Kernel)
Validation Accuracy	90.67%	93.83%
Training Time	Very Fast (< 1 sec)	Moderate (~5–10 sec)
Inference Speed	Fast	Fast
Memory Usage	High (stores all data)	Low (only support vectors)
Sensitivity to Noise	High	Medium

Key Observations

- SVM Outperformed k-NN: The SVM with RBF kernel achieved significantly higher accuracy (93.83% vs. 90.67%) due to its ability to learn complex, non-linear decision boundaries — crucial for distinguishing visually similar materials like plastic and glass.
- k-NN Was Simpler but Less Accurate: While k-NN required no iterative optimization and was faster to train.

Conclusion

Based on the superior validation accuracy and robustness to feature noise, the SVM model was selected as the final deployed model for the real-time application. However, k-NN remains a viable alternative for scenarios requiring minimal training time and interpretable predictions, especially when computational resources are limited. This comparison highlights a fundamental trade-off in ML: simplicity and speed (k-NN) vs. accuracy and generalization (SVM). For this project's goal of achieving ≥85% accuracy, SVM proved to be the more suitable choice.

Phase4: Rejection Mechanism for “Unknown” Class

We designed a probabilistic rejection framework that ensures robustness against out-of-distribution or ambiguous inputs (e.g., blurred images, foreign objects, or materials not in the training set).

4.1. Unified Confidence Metric

Both classifiers (SVM and k-NN) were configured to output class probabilities via the predict_proba() method:

- For k-NN (with weights='distance'), predict_proba() returns normalized inverse-distance-weighted votes, where closer neighbors contribute more to the final probability.
- For SVM, we enabled probability=True during training, which applies Platt scaling to convert decision function outputs into calibrated probabilities.

This unified output allows us to define model confidence identically for both architectures:

Confidence = max(predicted_class_probabilities)

4.2. Threshold Calibration and Rejection Rule

We calibrated the rejection threshold on the validation set to balance known-class accuracy and rejection rate. A threshold of 0.7 was selected because it:

Achieves >88% accuracy on non-rejected samples,

Maintains a rejection rate below 12% on in-distribution validation data,

Effectively rejects out-of-distribution inputs (e.g., non-waste objects) during real-time testing.

This satisfies the requirement that class ID 6 represents uncertain or unrecognizable inputs.

4.3. Integration and Deployment

The rejection logic is encapsulated in a generic, model-agnostic function (unknownlogic.py) that works with any scikit-learn classifier supporting predict_proba(). This design:

Ensures fair comparison between SVM and k-NN (same confidence metric),

Simplifies real-time deployment (one inference pipeline),

Guarantees that the competition submission (model.pkl) includes all components needed to produce ID 6 predictions on the Hidden Test Set.

Phase5: Real-Time System Deployment

The Python script implements a feature extraction utility that prepares image data for material classification by converting raw images or live camera frames into numerical feature vectors using a pre-trained model. The main functionalities and workflow of the script are as follows:

Importing Required Libraries

- OpenCV (cv2): For image capture, processing, and displaying images.
- NumPy (np): For handling numerical data, such as feature vectors.
- OS and sys: For file path management and dynamically adding project directories to the Python path.
- Joblib: for efficiently saving and loading Python objects
- Custom Modules:
    - Unknow_logic: it used for prediction with confidence-based rejection and to handle the use of different models inside the real time application

Path Setup

This code sets up the project root directory and adds it to Python's module search path to enable imports like feature_extraction. It then checks for and loads a trained SVM model pipeline (including the model, scaler, class labels, and rejection threshold) from a pickle file. If the model file is missing, it prints an error and exits, prompting the user to train the model first.

Frame Classification Function

Using this function to be used whether in live camera or using image path :

- Accepts a frame, and handle if it was corrupted
- Extracts features using extract_features_from_frame.
- Process the features to be used in prediction
- Uses the SVM model to predict the material category.
- Also Extract the confidence level of the model for this frame
- Returns : prediction label and confidence level

Main Application Workflow **def main():**

1. **Choose how you want to try the model**

   Give the user the chance to choose between using live camera app or entering image file path to try the application

2. **Live Camera Mode**
   a. Open the default webcam using OpenCV.
   b. Capture frames continuously in a loop.
   c. Display a "press q to quit" message in the bottom-right corner.
   d. Classify each frame in real-time and show the label and confidence level.
   e. Stops when the user presses 'q' or if the camera fails.
   f. Releases the video capture object to free the camera or video file.
   g. Releases the video capture object to free the camera or video file.

3. **Image File Mode**
   a. Prompts the user for the path of an image file.
   b. Loads the image and applies the classification.
   c. Displays the predicted label in the console and the confidence label.
4. **Loop and Exit**

   The loop of user entering the frame and the model predict and output the prediction label and confidence level is continuing until the user press (3)

References

1. https://www.lightly.ai/blog/data-augmentation
2. https://www.kaggle.com/code/naim99/data-augmentation-techniques
3. https://medium.com/data-science/exploring-feature-extraction-with-cnns-345125cefc9a
4. https://docs.opencv.org/4.x/db/deb/tutorial_display_image.html