Real-Time Scrap Classifier & Robotic Pick Simulation

1. Problem Understanding

In industries like recycling and waste management, one of the biggest challenges is **automating the identification and segregation of scrap materials**. Manual sorting is not only time-consuming but also inconsistent and costly. The goal of this project was to build a **real-time scrap classification system** that can recognize different categories of scrap material and simulate how a robotic arm would pick them up for efficient recycling.

---

2. Step-by-Step Breakdown

- **Dataset Preparation:**
  I collected and organized a dataset consisting of multiple categories of scrap (plastic, metal, glass, paper, etc.). Initially, the dataset was in a **folder-in-folder structure** with thousands of images spread across subfolders. I preprocessed it by restructuring, labeling, resizing, and augmenting the images to improve model generalization.

- **Model Training:**
  I trained a **Convolutional Neural Network (CNN)** for image classification. Transfer learning with pretrained models (like ResNet) was explored to improve accuracy. The model was evaluated on training/validation/test splits and tuned using techniques like learning rate scheduling and dropout.

- **Real-Time Integration:**
  Using **OpenCV**, I integrated the trained model into a **real-time video pipeline** to classify scrap objects live.
  To simulate an industrial setup, I designed a **robotic pick mechanism** that responds to the classification results — essentially showing which scrap item would be picked and sorted.

- **Deployment:**
  The solution was deployed with **Streamlit**, providing a simple and interactive web interface where users can upload images, view predictions, or test the model on live video streams.

---

3. Key Decisions

- Opted for **transfer learning** instead of training from scratch to save time and leverage pretrained image representations.

- Chose **Streamlit** for quick deployment and demo purposes due to its ease of use and clean UI.

- Designed the pipeline to be modular so that it can later be integrated with **robotic hardware** for real-world automation.

---

4. Challenges and Learnings

- **Dataset structuring:** Initially, the dataset appeared disorganized, which required significant effort in cleaning and labeling.
- **Class imbalance:** Some scrap categories had fewer images, so I used **data augmentation** to balance them.
- **Real-time performance:** Running inference on video streams needed optimization; I learned how to manage **frame skipping and batch predictions** for smoother outputs.
- **Robotic simulation:** Since I didn't have access to an actual robotic arm, I simulated the process, which gave me insights into how vision and robotics interact.

These challenges helped me appreciate the importance of **data quality, optimization techniques, and practical deployment constraints**.

---

5. Optimizing for Edge Deployment

If deployed on an **edge device** (like a Raspberry Pi or Jetson Nano), I would:

- Use a **lightweight CNN model** (e.g., MobileNet or EfficientNet-lite) instead of heavy architectures.
- Apply **quantization and pruning** to reduce model size and improve inference speed.
- Use **TensorRT or ONNX runtime** for faster deployment on embedded GPUs.
- Minimize frame rate processing (e.g., classify every 2nd frame) to balance accuracy and latency.