ECE 6380 AI Hardware

Course Project Final Report

Topic: Edge-AI Waste Classification on OpenMV H7

Instructor: Mircea R. Stan

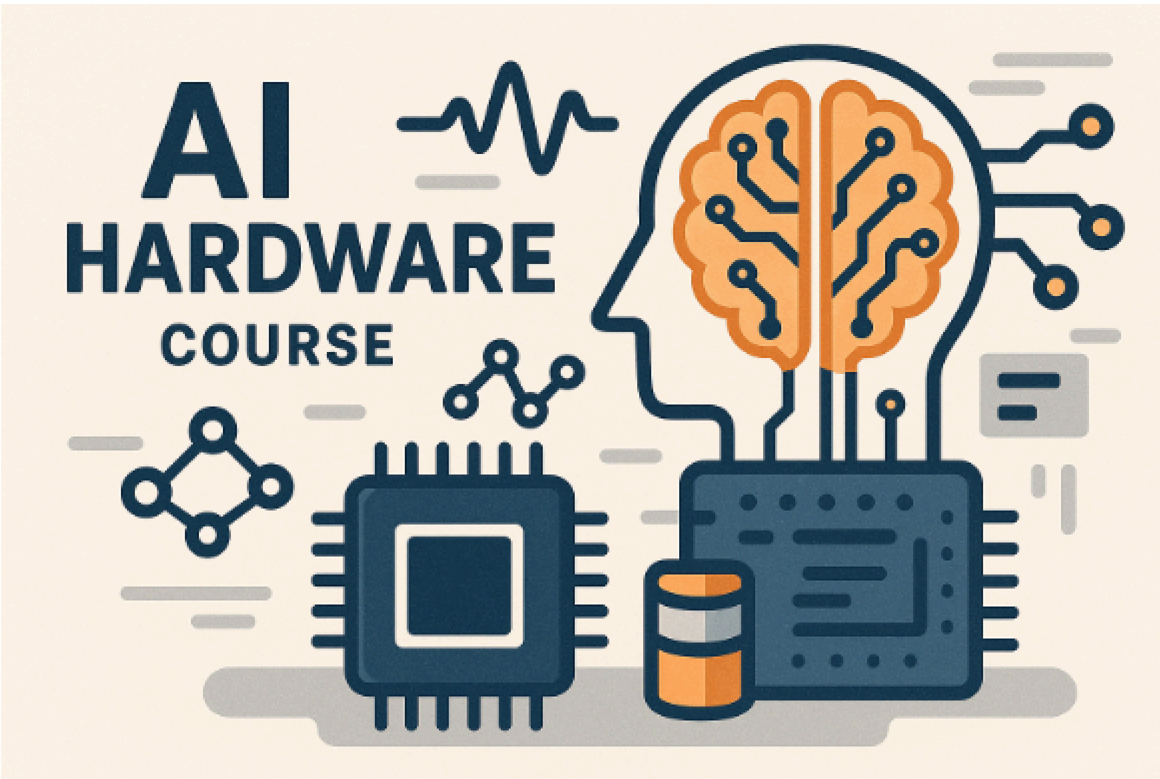
Author 1: Mengzi Cheng (cdd6yg)

Author 2: Shuai Tu (bjb4hf)

Author 3: Sirui You (jxb2wz)

Author 4: Xueyi Zhang (fyx9fy)

Date: DEC 19 - 2025



1. Abstract

This AI Hardware project looks into how far we can take a low-power microcontroller board for real-world computer vision by making a waste-image classifier that only works with an OpenMV H7 camera module. We want to be able to tell what kind of material a piece of trash is made of—glass, metal, paper, or plastic—just by looking at one picture. All of the processing will happen on the device itself, without any connection to the cloud. We use TrashNet as our main dataset and clean the images offline. Then we use Edge Impulse to build an end-to-end pipeline that includes data upload, preprocessing (resizing and converting to grayscale), a transfer-learning model based on MobileNetV2 96×96 0.35, training, and deployment as a TensorFlow-Lite model for OpenMV. The trained model has a validation accuracy of about 72.1%, a loss of 0.66, and an AUC of 0.92. The easiest class is paper, and the hardest is glass/plastic. A single inference on the device takes about 36 milliseconds and uses about 215 KB of RAM and 536 KB of flash memory. This makes it possible for the MCU to run in real time. We put the exported trained.tflite and labels.txt files into an OpenMV Python script that captures video from the camera all the time, smooths over recent frames with a majority vote, and adds predicted labels and confidence to the live image. Our tests show that the prototype can accurately identify simple objects like paper tissue or glasses frames in real time. This is a real-world example of how to use vision models on small edge hardware.

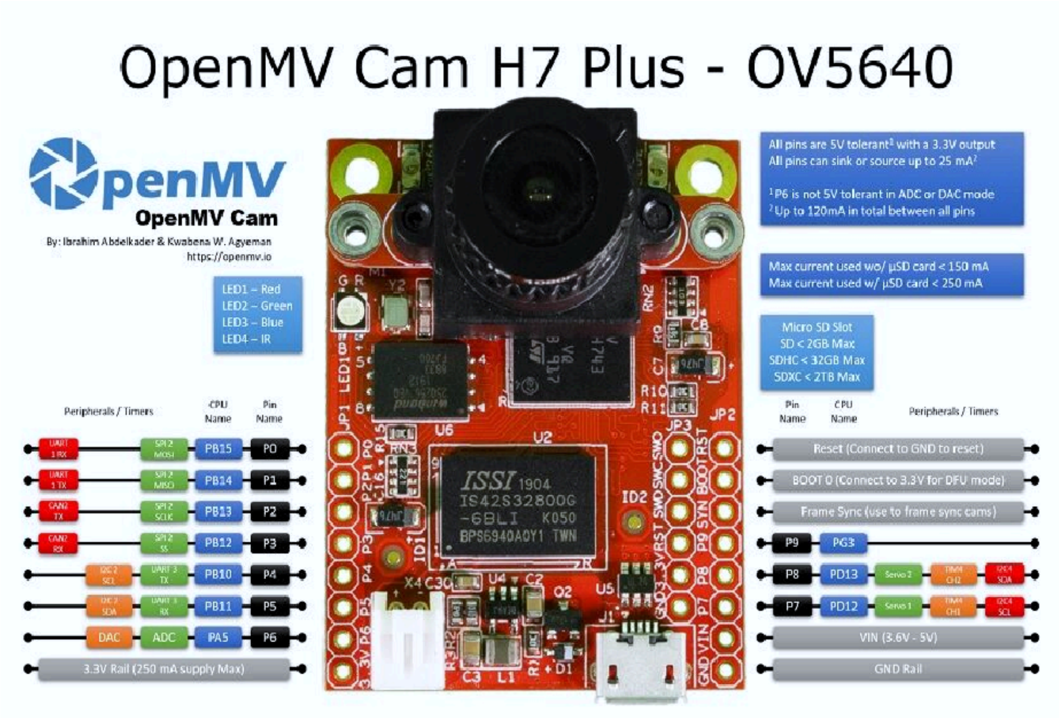
2. Motivation&Goals

A lot of cities are moving toward making people sort their trash. But sorting by hand takes a lot of time, is prone to mistakes, and is often hard for people who aren't experts to understand. Computer vision-based automatic waste classification can help users, raise recycling rates, and cut down on contamination in recycling streams.

Most current systems use cloud-based inference, which means that a camera takes pictures, sends them to a remote server, and a large model processes them. This results in a number of issues:

- Latency: The round-trip time might be too high for interactive feedback at the bin.
- Connectivity: Not every place has Wi-Fi or cell phone coverage.
- Privacy: sending raw pictures of the environment to the cloud is worrisome to people.

For this project, we look into a different way: running the classifier directly on a small MCU board with a built-in camera (OpenMV H7).



Edge inference makes it possible to:

- Low cost and low power because all you need is a microcontroller.
- Works on its own, without needing a network.
- Better privacy because the pictures never leave the device.

So, we have two reasons for doing this: (1) to make a useful demo of edge-AI waste classification, and (2) to learn about the trade-offs between model accuracy, speed, and resource use when using machine learning on limited hardware.

3. Related Work

3.1 Vision-based Waste Classification

Most of the time, deep learning is used to classify waste as an image classification problem. The model uses one RGB image of an object in fairly controlled lighting to guess what kind of trash it is (for example, paper or plastic). Most of the time, previous work in this area has used CNNs, like ResNet or MobileNet, trained on datasets like TrashNet, which has labeled pictures of different types of trash taken with regular cameras.

Models with millions of parameters are common on desktop and cloud platforms. But on microcontrollers, memory and processing power are very limited, so we need to use TinyML techniques and lightweight architectures.

3.2 TinyML and Edge Impulse

TinyML's main goal is to bring machine learning to microcontrollers that only have tens or hundreds of kilobytes of RAM. TinyML models need to be heavily compressed and optimized so that they don't need GPUs and gigabytes of memory. This is often done with quantization and compact backbones like MobileNet. Edge Impulse is a cloud-based platform for making TinyML apps.
It lets you collect and label data:

- Blocks for processing signals, like resizing images or using MFCC for audio
- Pipelines for training classification models
- Deployment options for a lot of boards, like Arduino and OpenMV H7 devices

Edge Impulse is the main tool we use to connect the dataset, model design, training, and deployment in our work.

4. Problem Definition

Our job is to look at a single picture taken by the OpenMV H7 camera and figure out what kind of waste it is: glass, metal, paper, or plastic. Then, on the device, show the prediction and confidence in real time.
Important limits are:

- The model must not use more memory (RAM and flash) than OpenMV H7 can handle.
- The time it takes to make an inference must be short enough for performance that is almost real-time.
- The entire pipeline, from capturing an image to making a prediction, must work completely on the device, without any cloud resources.

We view this as a multi-label classification problem, having four labels that cannot be the same.

5. Dataset & Preprocessing

5.1 Dataset Source

We used the TrashNet dataset containing images of household waste collected with iPhone cameras in normal lighting conditions. So, in this project, we only care about four materials:

- Glass
- Metal
- Paper
- Plastic

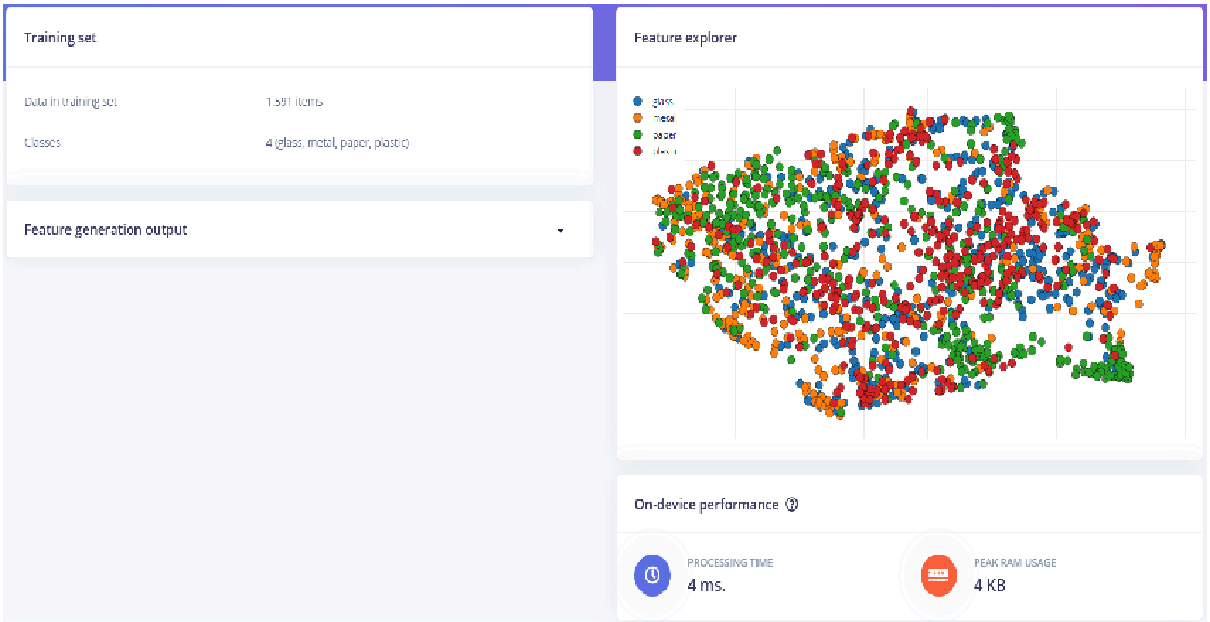
All images are consecrated in RGB scale, and the resolution is 512×384 pixels. Final filtered Edge Impulse dataset: 1,984 images. The challenge is to classify a single image in one of the four classes.

5.2 Offline Preprocessing

Before uploading to Edge Impulse, we do some basic preprocessing on a laptop, which is not connected to the internet:

- Ensure that all images are RGB images and fairly consistent in resolution (512×384).
- Folder structure: Create a directory per type (glass, metal, paper, plastic) to make labeling images simple.
- Quality control: Localize and remove any clearly broken images and mislabeling.

This results in a dataset that is easy to import.



5.3 Online Preprocessing in Edge Impulse

We used an additional preprocessing pipeline in Edge Impulse that is a good match for embedded inference:

- Image rescaling: Rescale and center crop all images to 96×96, while trying to keep the original aspect ratio.
- Grayscale: Changes the color space of the image to grayscale to reduce the number of channels used and save memory.
- Feature vector: Each grayscale image is turned into a small feature vector that the neural network uses as input.
- Performance: Edge Impulse says that this step of preprocessing takes about 4 ms to process and uses about 4 KB of RAM per image at its peak.

The result is a small but useful representation that works well with the OpenMV H7.

6. Model & Training

6.1 Hardware and Software Platforms

Hardware: OpenMV Cam H7 Plus

The OpenMV Cam H7 Plus is our target inference platform. Its main specs (from the board datasheet and our slide summary) are:

- MCU: ARM Cortex-M7, STM32H743II
- Clock speed: 480 MHz
- Memory: 34 MB of flash memory and 33 MB of external RAM
- The camera sensor is an OV5640 with 5 MP and a resolution of up to 2592×1944.
- F2.0, M12 mount, and 2.8 mm lens
- USB FS (12 Mbps) and a microSD slot for connecting
- Power: 3.6–5 V in, active current is about 230–240 mA at 3.3 V

This board is a good size and has a lot of processing power, making it a good choice for edge-vision tasks.

Software Stack: We depend on two main pieces of software.

- Edge Impulse Studio is a web-based tool for managing datasets, preprocessing them, training models, and exporting a TFLite model that is specifically made for OpenMV.
- OpenMV IDE: a desktop IDE for writing MicroPython scripts, moving files to the board, and watching serial output.

6.2 Project Pipeline and Team Roles

The slides' "Pipeline & Team Responsibilities" figure shows how our workflow works: dataset → split and upload → preprocessing → training → exporting and deploying → live inference.

- Mengzi Cheng (Team Leader): planning and coordinating the project; overseeing the integration; and organizing the presentation.
- Shuai Tu (Hardware & Deployment): Setting up the OpenMV H7 and the camera; deploying the model; and supporting demo hardware.
- Sirui You (Software and Integration): OpenMV scripting, inference pipeline, debugging, and serial/logging integration.
- Xueyi Zhang (Model & Documentation): getting the dataset ready, training and testing the model, and PPT designing and reports.

This division of labor lets people work on different parts of the pipeline at the same time while keeping everything in sync.

6.3 Model Architecture and Training

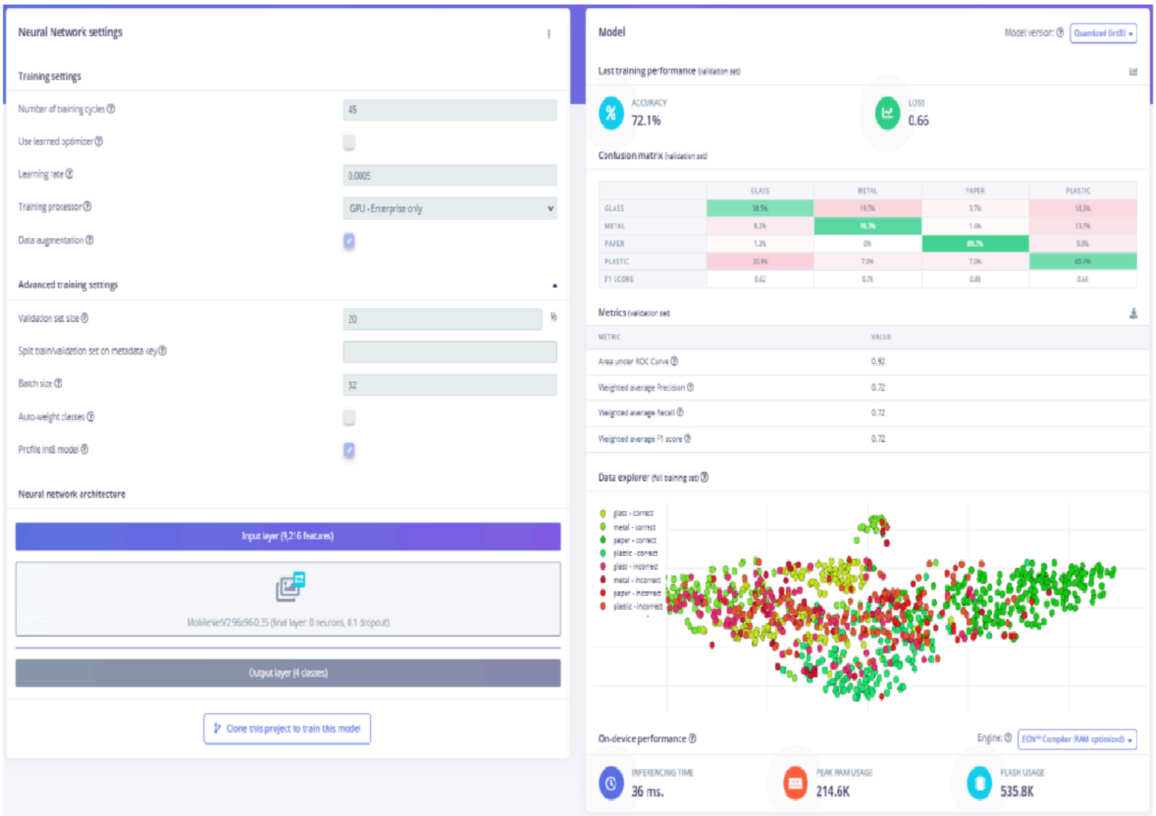
The "Transfer Learning (Images)" block in Edge Impulse is the part of the impulse that learns. The block takes a 96×96 grayscale image as input and gives four probabilities as output: glass, metal, paper, and plastic.

Backbone and classifier:

- Backbone: MobileNetV2 96×96 0.35, a lightweight CNN made for devices which are built in.
- Dropout: 0.1 before the last classification layer to stop overfitting.
- Output layer: a fully connected layer with 4 units and a softmax activation.

Hyperparameters for training:

- 45 training cycles (epochs)
- Size of the batch: 32
- Rate of learning: 0.0005
- Validation split: 20% of the data is set aside for validation.
- Data augmentation: on (random changes to make it more robust)



6.4 Model Deployment to OpenMV

When the validation performance is good enough, we use Edge Impulse's OpenMV deployment option to export the trained model. The export package has:

- trained.tflite is a quantized TensorFlow Lite model, and labels.txt is a list of class names in the right order.

Next, we:

1. Get the firmware package that was made.
2. Use a USB cable to connect the OpenMV H7 to the PC.
3. Use OpenMV IDE to copy trained.tflite and labels.txt to the board's internal storage.
4. Write and upload a MicroPython script called main.py that loads the model and makes predictions.

6.5 On-Device Inference Script

After training and exporting the model from Edge Impulse as “trained.tflite” plus “labels.txt”, we run all inference directly on the OpenMV H7 with a MicroPython script (main.py). The script follows the “on-device real-time inference pipeline” shown in the presentation: setup, per-frame inference, sliding-window voting, and final display.

Camera and model initialization

At the beginning of the script, we configure the camera and the runtime environment:

- Reset the sensor and set the pixel format to RGB565.
- Use QVGA (320×240) as the frame size and then apply a 240×240 window so the input roughly matches the 96×96 model input after Edge Impulse’s internal resize.
- Call sensor.skip_frames(time=2000) to allow auto-exposure and white-balance to settle before inference starts.

Then we load the model and labels:

“net = ml.Model("trained.tflite",

load_to_fb=uos.stat('trained.tflite')[6] >

(gc.mem_free() - (64 * 1024)))

labels = [line.rstrip('\n') for line in open("labels.txt")]”

The “load_to_fb” flag is chosen by comparing the “.tflite” file size with the available free memory minus a 64 KB safety margin. This prevents out-of-memory errors on the microcontroller. If either file is missing, the script raises a clear exception message so the user knows to copy both the model and label files onto the board.

Per-frame inference and raw output

Inside the main loop, the script captures a new frame and runs inference:

“img = sensor.snapshot()

scores = net.predict([img])[0].flatten().tolist()

predictions_list = list(zip(labels, scores))”

For each frame, it prints all label–score pairs and the current clock.fps() value to the serial console. This gives a “raw” view of the model output and the actual speed in frames per second on the OpenMV H7. The script also finds the best label for this frame by taking the index of the maximum score (scores.index(max(scores))).

Sliding-window majority vote

Single-frame predictions can flicker when the image is noisy or near a decision boundary, so we smooth them with a sliding-window majority vote. We keep two lists, vote_labels and vote_scores, that store the best label and its confidence for the most recent frames:

```
“VOTE_WINDOW = 15

vote_labels.append(best_label)

vote_scores.append(best_score)

if len(vote_labels) > VOTE_WINDOW:

    vote_labels.pop(0)

    vote_scores.pop(0)”
```

To get the final label for display, the script counts how often each label appears in the window and selects the one with the highest count. It then computes the average score of that label over all frames in the window, giving a more stable confidence value:

```
“for lab in set(vote_labels):

    c = vote_labels.count(lab)

    ...

txt = "%s %.2f" % (majority_label, majority_score)

img.draw_string(5, 5, txt, color=(255, 0, 0), scale=2)

print("VOTE ->", majority_label, "avg_score =", majority_score)”
```

This produces a real-time display, such as paper 0.95 in the top-left corner of the camera view, while the serial terminal shows both the per-frame probabilities and the stabilized voted result. Together with the Edge Impulse preprocessing and MobileNetV2 model, this script implements a complete on-device waste-classification pipeline that runs fully on the OpenMV H7 without any cloud support.

7. Experimental Results

7.1 Results of the Edge Impulse Validation

We first test the model inside Edge Impulse by using 20% of TrashNet as a validation set. With the MobileNetV2 96×96 0.35 backbone and data augmentation turned on, the model does the following:

- 72.1% correct
- Loss: 0.66
- AUC: 0.92
- Weighted precision, recall, and F1 are all about 0.72.

We can see from the confusion matrix on this split that:

- The class on paper is the easiest, with a recall rate of almost 90%.
- Glass and plastic are harder to tell apart because they look similar and reflect light strongly.
- Metal is in the middle of the two groups. It has decent precision and recall, but it still gets confused with other shiny things.

Edge Impulse also gives the OpenMV H7 an estimate of how well it will work on the device:

- Time to make an inference: about 36 ms per image
- Maximum RAM usage: about 215 KB
- Flash usage: about 536 KB

These numbers imply that the model is likely to fit in our board and has a decent speed to run at a few frames a second, even with all the overhead of Python.

7.2 On-Device Qualitative Tests

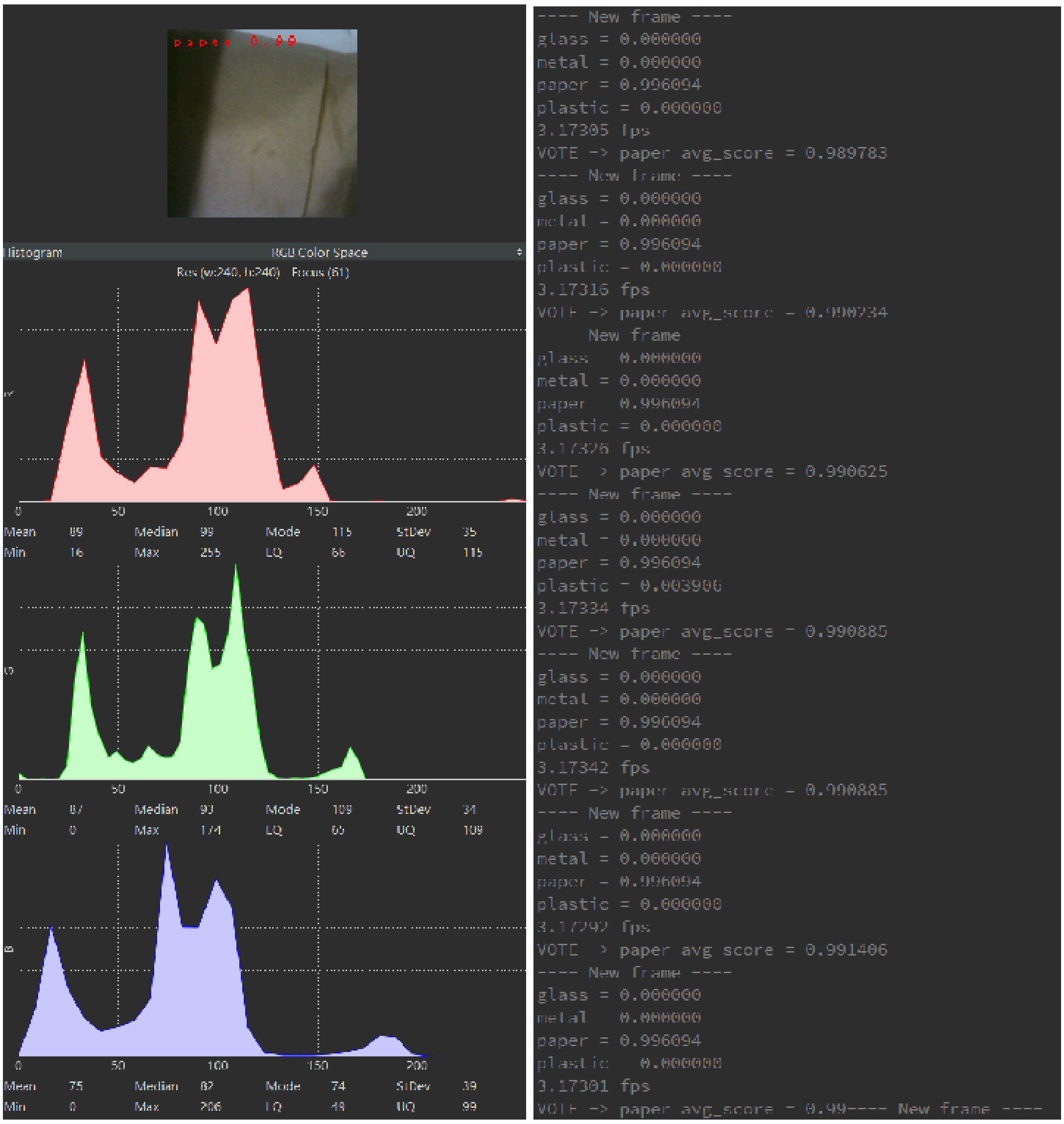
After the tests, we deploy to OpenMV H7 and try it out on-device in edge cases. The entire pipeline (camera acquisition, preprocessing, TFLite inference, best-majority voting, and overlay) is run on-device at 3.17 FPS across all tests.

Case 1: Paper

We put a folded paper tissue in front of the camera and switched on the interior lights. The final result was:

- Last label: paper
- Confidence level: about 0.99
- Speed: about 3.17 frames per second

The predictions snap to the right class in a few frames and are stable from thereon out. This was expected as paper is one of the less complicated classes.

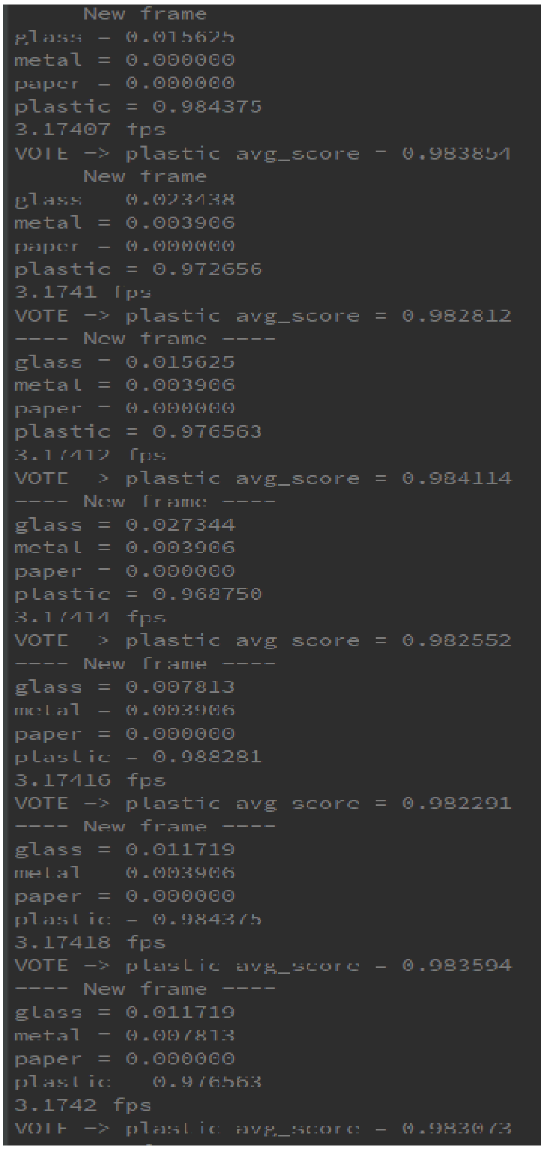


Case 2: Plastic

Next, we try a plastic lid. This is the output of the camera feed and majority voting stepping in:

- Last label: plastic
- Average trust: ≈ 0.99
- Speed: about 3.17 frames per second

The probabilities of plastic matter the most in all frames; majority voting acts as a smoothing filter. It is now possible to determine a majority of plastic items in real-time.

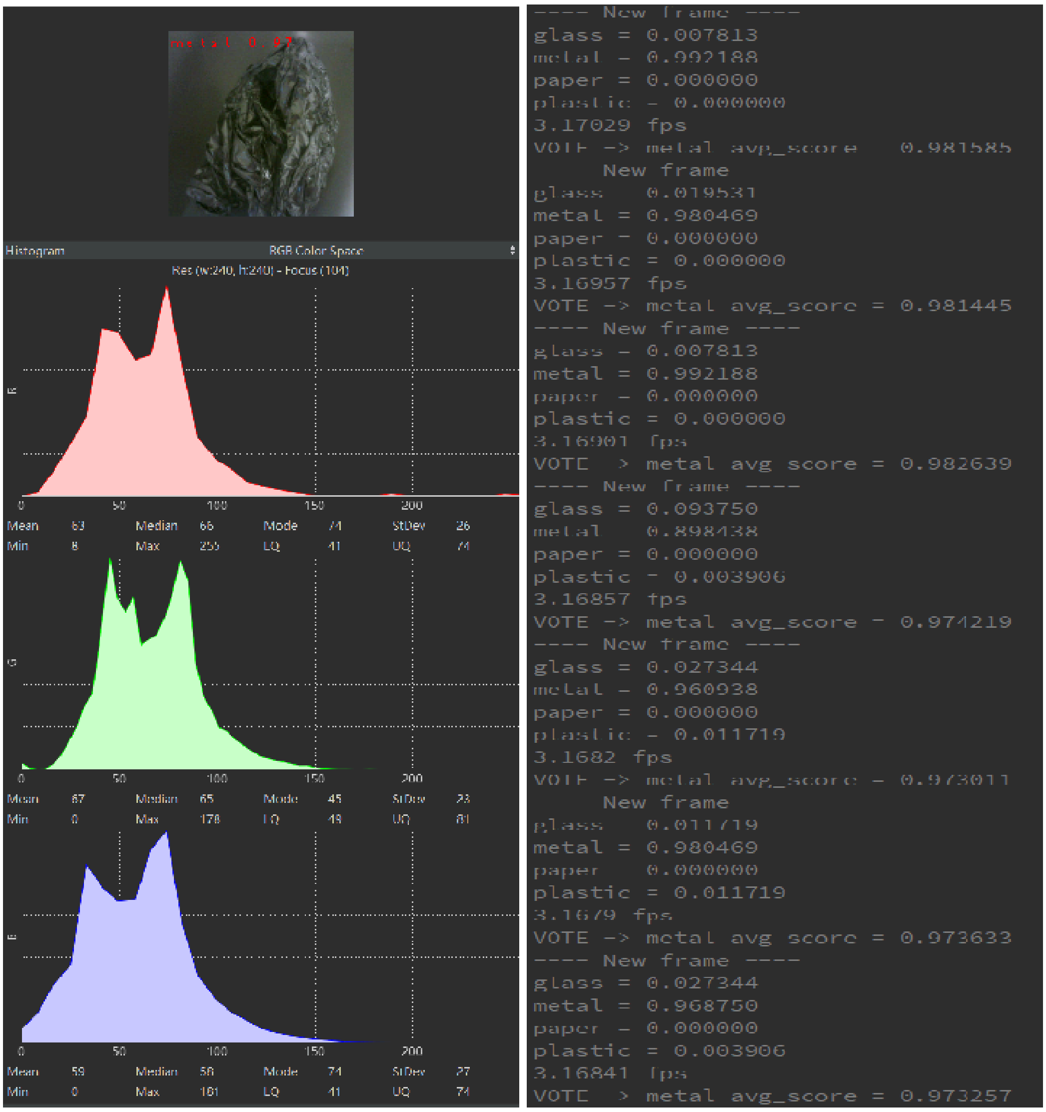


Case 3: Metal

We use a crumpled paper ball to try metal. This is more difficult because the blinking light has a strong influence.

- Last label: metal
- Confidence level: about 0.97
- Speed: 3.17 frames per second

Per-frame probabilities do fluctuate a bit more than previous tests, but the voted label is mostly stable and correct. This is good enough evidence to show that the model learns meaningful information regarding metal.

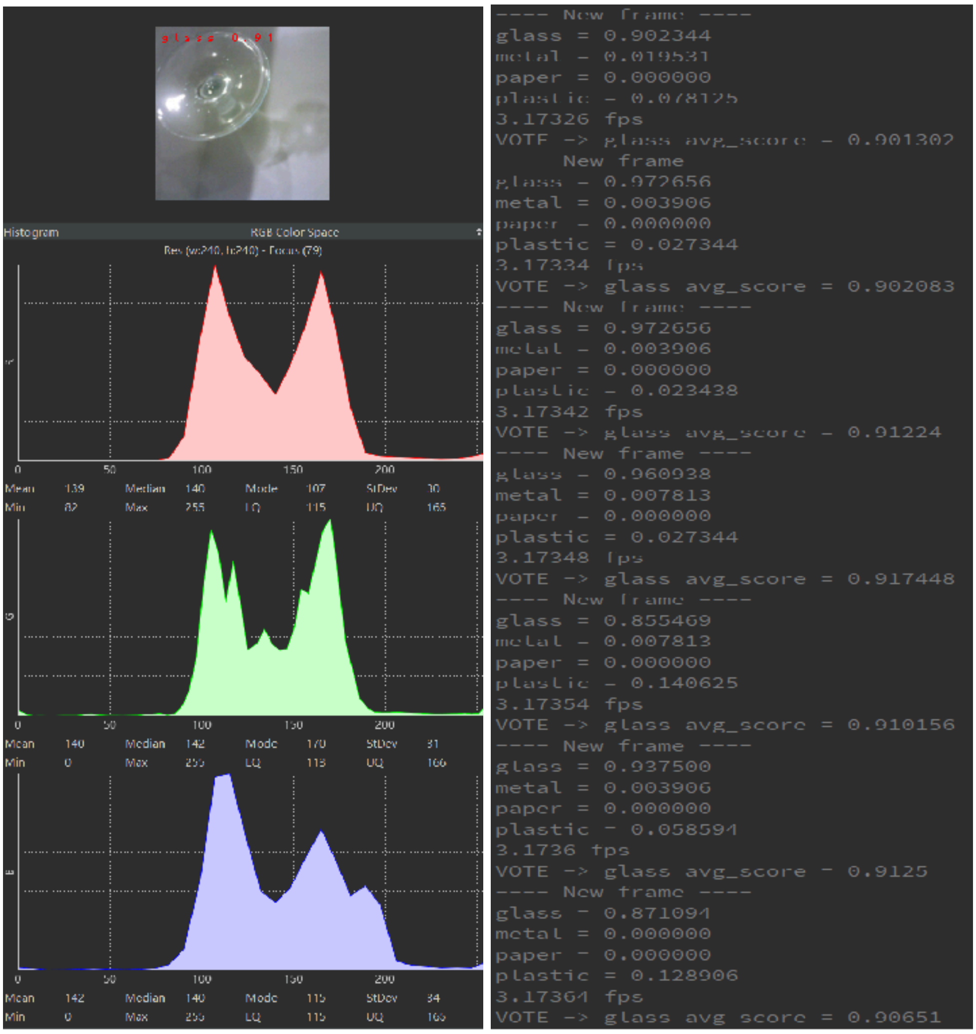


Case 4: Glass

Finally, we try a glass cup. Glass has reflections and is somewhat transparent, while also altering background.

- Final label: glass
- Average level of confidence: 0.91
- Speed: ~3.17 FPS

In this case, per-frame probabilities are more unstable when cup or lighting changes. Our sliding-window voting method was able to avoid majority-vote label from changing while still being able to output the right label most of the time.



7.3 Discussion

The system makes correct majority-vote labels at about 3.17 FPS in all four on-device demos. This shows that real-time waste classification is possible on the OpenMV H7. But the tests also show a clear pattern:

- The most stable and accurate classes are usually paper and plastic. It's easier to tell them apart from the background, so the model converges quickly and stays very confident.

- Metal changes more from frame to frame because its look changes with the angle and lighting, but the model still correctly identifies it in our tests.

- The hardest group is glass. The predicted probabilities change because of things like transparency, background colors, and lighting. The model relies heavily on majority voting to keep the label the same.

Even with these problems, the combination of a small CNNs, Edge Impulse preprocessing, and the sliding-window voting scheme gives useful predictions in a lot of real-world situations. It also sets a good standard for future improvements in data collection and model tuning on MCU hardware.

8. Limitations

Our current prototype has major limitations:

1) The dataset and deployment domain do not match.

The model is trained on images taken in specific lighting and backgrounds on phones. The snapshots are of a different resolution, field of view, and capture conditions on OpenMV H7. When we move from the pristine dataset validation images to real objects in front of our camera, the difference creates an unreliable system.

2) Classes of different difficulty.

Users have a much higher confidence predicting paper and plastic than predicting metal and glass. The appearance of metallic and glass objects is heavily affected by environment reflections and object transparency and background colors. This increases the odds and makes the error distribution less sparse.

3) Real-time and resource constraints are very aggressive.

Full pipeline on the OpenMV H7, from camera image capture, preprocessing, TFLite inference, majority vote, to overlaying predictions, is ~3.17 FPS. This is reasonable for our experiment, but leaves no room for complex processing or a better model.

4) Coverage in evaluations is limited.

Our device runs on a quick set of common household objects and scenes at home. We lack a comprehensive, diverse validation set collected in a systematic way with a variety of scenes, lighting, and camera angles.

9. Future Work

To get around these problems and make the system more reliable, we plan to add the following:

1) Get OpenMV data from within the domain.

Use the OpenMV camera to make a new dataset that includes a variety of backgrounds, distances, and lighting setups. Training (or fine-tuning) on this data from the same domain should close the gap between domains and make it more reliable in real life.

2) Make things work better on glass and metal.

Make sure that metal and glass get more attention during training by designing targeted data augmentation for reflective and transparent objects (for example, by making brightness and contrast changes stronger, adding blur, or adding background noise). You could also try class-balanced sampling or loss functions.

3) Make the pipeline on the device as good as it can be.

To raise the FPS above the current ~3.17 FPS, profile the latency of each stage (capture, preprocessing, inference, display/printing) and get rid of bottlenecks. For instance, you could cut down on verbose serial printing, make overlays easier, or make the frame size smaller.

4) Look into ways to improve the model.

To use less memory and speed up inference, try even lighter backbones or more aggressive quantization schemes. If possible, keep accuracy close to the current baseline. Increase testing, and make the explanation easier to understand for non-experts.

5) Increase testing and improve user experience.

Run tests in the field at labs, and cafeteria environments, efficiently simulate recycling bins. Improve signals through LEDs, sound, and icons that are more meaningful for non-expert users.

10. Conclusion

We built a complete edge-AI waste sorting classifier solution that's designed to run on the OpenMV H7 microcontroller board. We have a TinyML pipeline using TrashNet dataset and Edge Impulse platform powered by image pre-processing, transfer learning with MobileNetV2 96×96 0.35 and deployment as a TF-Lite model on OpenMV C++ microcontroller firmware. Our classifier is capable of ~72.1% validating accuracy in four classes (glass, metal, paper, plastic), within the RAM (~215 KB) and flash (~536 KB) size constraints of the OpenMV MCUs.

Our unified python program on the device for efficient camera image gathering, model inference, sliding-window majority votes, and predictions overlay on top of image for consistent, human-interpretable outputs for common cups, lids, foil balls, paper tissues. The prototype demonstrates on a clear trajectory that valuable dumping image classification can run on low-cost, small-footprint unmanned Hardware MCUs with limited testing and challenging metal and glass classes. This work plants a detailed roadmap of subsequent future efforts ranging from better data to model and MCU-centric optimizations, and the effect of decisions at each stage of processing pipeline on the AI-assisted user experience.

11. References

[1] aniass, "Waste-Classification," GitHub repository, available at: https://github.com/aniass/Waste-Classification

[2] Kemalkilicaslan, "Garbage Classification with Convolutional Neural Network (CNN)," GitHub repository, available at: https://github.com/kemalkilicaslan/Garbage-Classification-with-Convolutional-Neural-Network-CNN

[3] vasantvohra, "TrashNet – Deep Learning based Waste Segregation Project," GitHub repository, available at: https://github.com/vasantvohra/TrashNet