ECE 6380 AI Hardware

Course Project Final Report

Topic: Edge-AI Waste Classification on OpenMV H7

Instructor: Mircea R. Stan

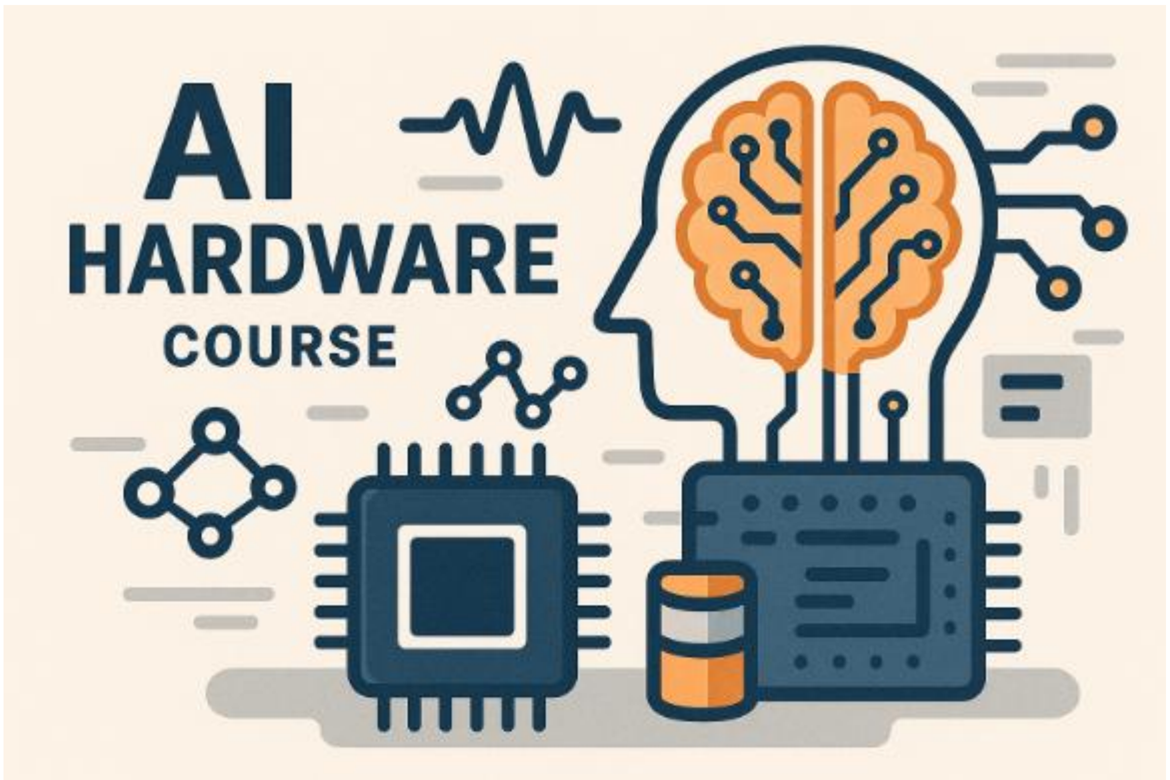
Author 1: Mengzi Cheng (cdd6yg)

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.

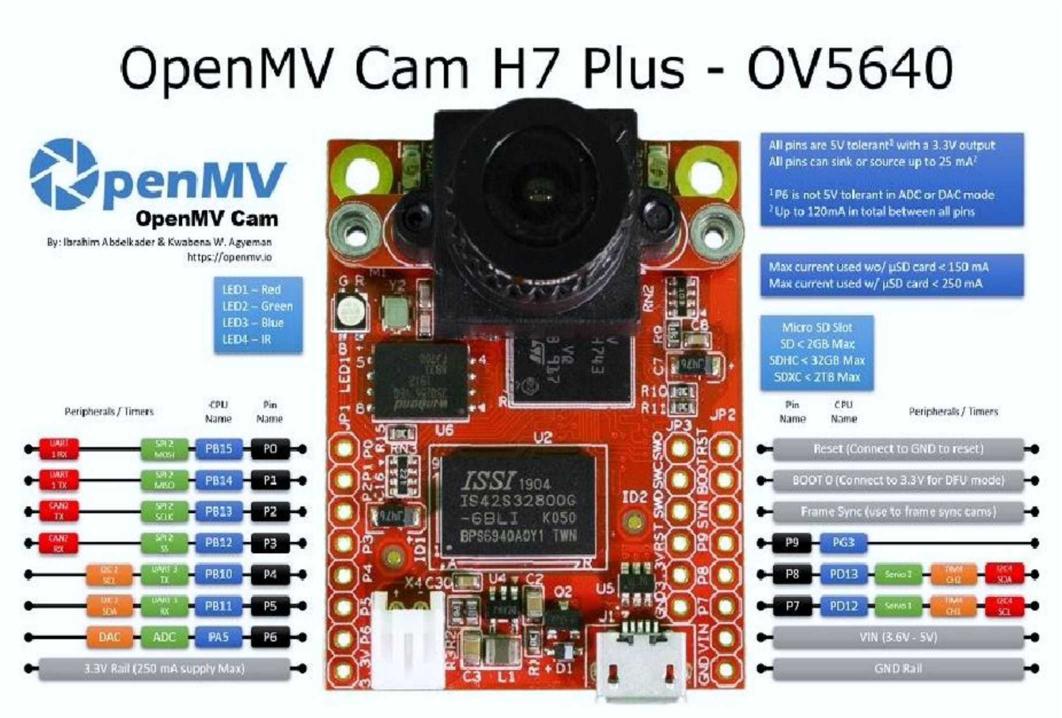
2. Motivation&Goals

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 class ification can help users, raise recycling rates, and cut down on contamination i n recycling streams.

Most current systems use cloud-based inference, which means that a camera t akes pictures, sends them to a remote server, and a large model processes the m. This creates a number of problems:

- Latency: The round-trip delay might be too long for interactive guidanc e at the bin.
- Connectivity: Not all places have Wi-Fi or cell phone service.
- Privacy: sending unedited pictures of the environment to the cloud can make people worry.

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).


OpenMV Cam H7 Plus - OV5640
OpenMV
OpenMV Cam
By: Ibrahim Abdelkader & Kwabena W. Agyeman
https://openmv.io
LED1 – Red
LED2 – Green
LED3 – Blue
LED4 – IR
All pins are 5V tolerant with a 3.3V output
All pins can sink or source up to 25 mA
Max current used wo/ µSD card < 150 mA
Max current used w/ µSD card < 250 mA
Micro SD Slot
SD < 2GB Max
SDHC < 32GB Max
SDXC < 2TB Max
Reset (Connect to GND to reset)
BOOT 0 (Connect to 3.3V for DFU mode)
Frame Sync (use to frame sync cams)
VIN (3.6V - 5V)
GND Rail
3.3V Rail (250 mA supply Max)

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 accur acy, speed, and resource use when using machine learning on limited hardwar e.

3. Related Work

3.1 Vision-based Waste Classification

Most of the time, deep learning is used to classify waste as an image classifica tion problem. The model uses one RGB image of an object in fairly controlled l ighting 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 platfor ms. But on microcontrollers, memory and processing power are very limited, s o 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 me mory. This is often done with quantization and compact backbones like Mobil eNet. 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 au dio
- Pipelines for training classification models
- Deployment options for a lot of boards, like Arduino and OpenMV H7 d evices

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

4. Problem Definition

Our job is to look at a single picture taken by the OpenMV H7 camera and fig ure out what kind of waste it is: glass, metal, paper, or plastic. Then, on the de vice, 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 perfor mance that is almost real-time.
- The entire pipeline, from capturing an image to making a prediction, m ust work completely on the device, without any cloud resources.

We see this as a multi-class classification problem with four labels that can't b e the same.

5. Dataset & Preprocessing

5.1 Dataset Source

We use the TrashNet dataset, which contains images of household waste captured with iPhone cameras under normal room lighting. For this project, we focus on four material types:

- Glass
- Metal
- Paper
- Plastic

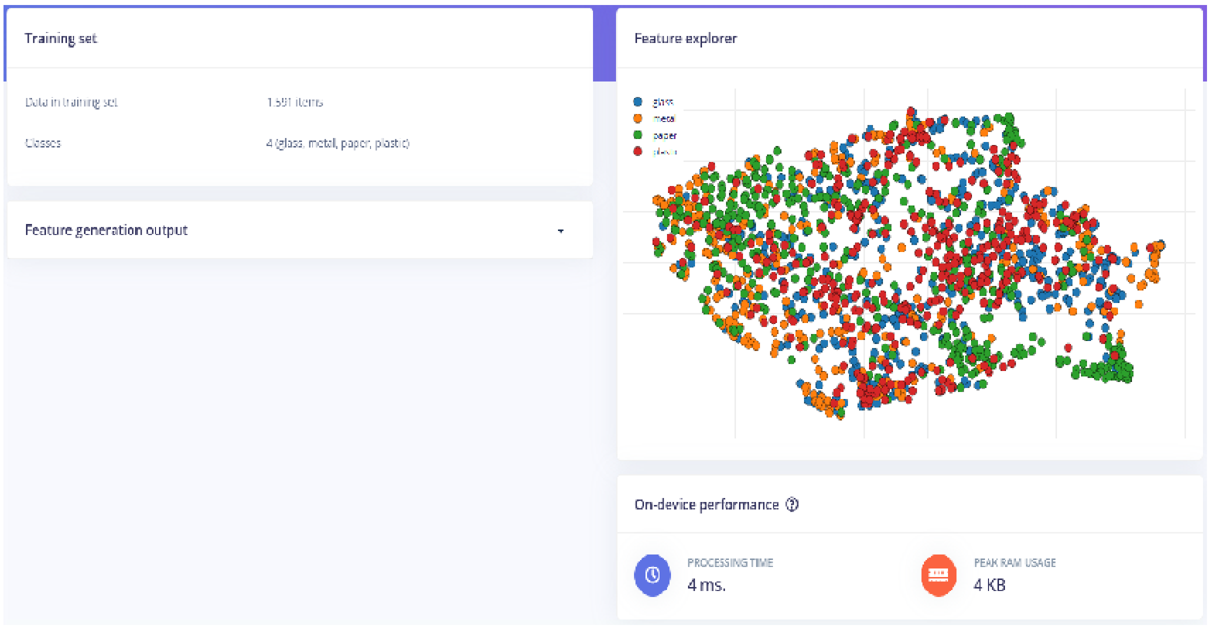
All images are in RGB and have a resolution of 512×384 pixels. After filtering, the Edge Impulse dataset has a total of 1,984 images. The job is to classify a single image into one of these four groups.

5.2 Offline Preprocessing

Before uploading to Edge Impulse, we do some simple preprocessing on a laptop that isn't connected to the internet:

- Make sure that all of the images are in RGB format and have the same resolution (512×384).
- Organizing your directories: put pictures in folders for each type of material (glass, metal, paper, plastic) to make labeling easy.
- Quality checks: when you find images that are clearly broken or mislabeled, take them out.

This step makes our dataset clean and organized so that we can import it.


Training set

Data in training set	1,591 items
Classes	4 (glass, metal, paper, plastic)

Feature generation output

Feature explorer

On-device performance

PROCESSING TIME 4 ms.

PEAK RAM USAGE 4 KB

5.3 Online Preprocessing in Edge Impulse

We use an extra preprocessing pipeline inside Edge Impulse that works well fo r embedded inference:

- Resize: All images are resized to 96×96 pixels in a way that keeps the ori ginal aspect ratio as much as possible.
- Color space: The resized image is changed to grayscale to use fewer cha nnels and less memory.
- Feature vector: Each grayscale image is turned into a small feature vecto r that the neural network uses as input.
- Performance: Edge Impulse says that this step of preprocessing takes ab out 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 OpenM V 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 pipeli ne, debugging, and serial/logging integration.
- Xueyi Zhang (Model & Documentation): getting the dataset ready, traini ng and testing the model, and making slides 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

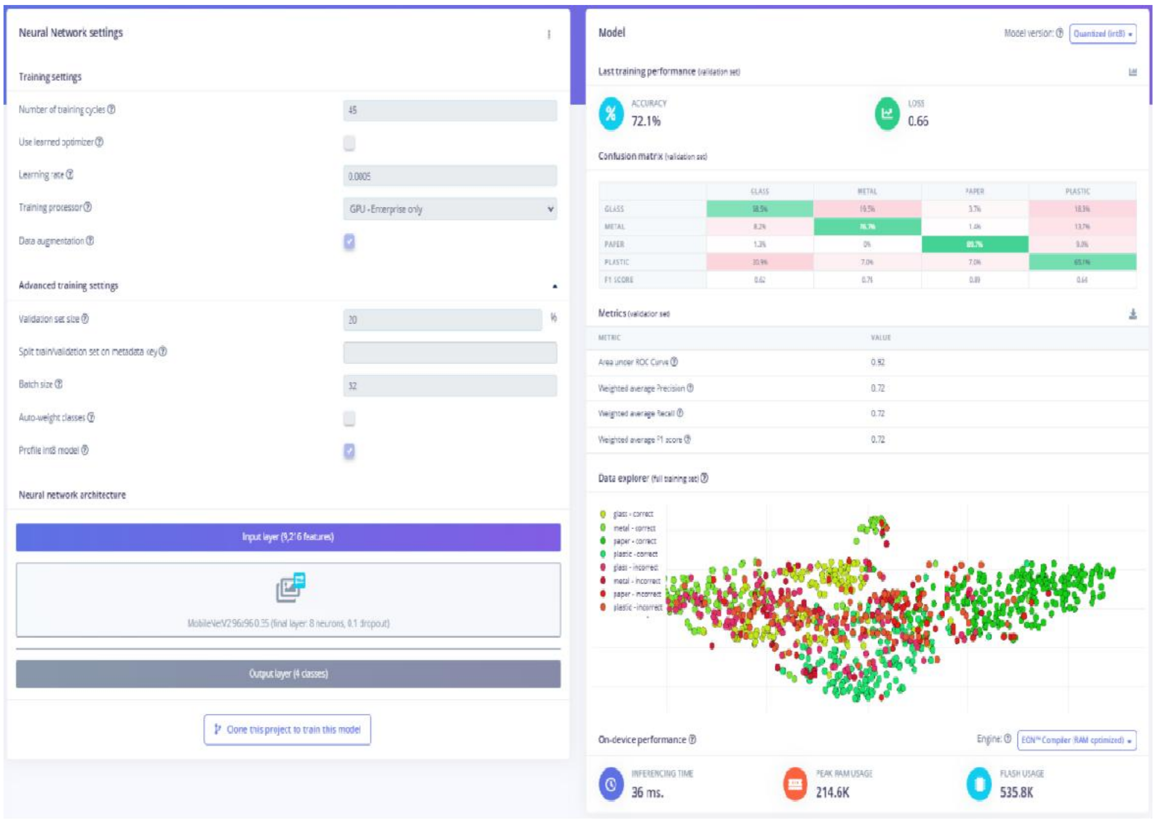
The "Transfer Learning (Images)" block in Edge Impulse is the part of the impu lse that learns. The block takes a 96×96 grayscale image as input and gives fo ur probabilities as output: glass, metal, paper, and plastic.

Backbone and classifier:

- Backbone: MobileNetV2 96×96 0.35, a lightweight CNN made for devic es 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 activatio n.

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)



Neural Network settings

Training settings

Number of training cycles ⓘ	45
Use learned optimizer ⓘ	☐
Learning rate ⓘ	0.0005
Training processor ⓘ	GPU - Enterprise only
Data augmentation ⓘ	✓

Advanced training settings

Validation set size ⓘ	20 %
Split train/validation set on metadata key ⓘ	
Batch size ⓘ	32
Auto-weight classes ⓘ	☐
Profile int8 model ⓘ	✓

Neural network architecture

Input layer (9,216 features)

MobileNetV2 96x96 0.35 (final layer: 8 neurons, 0.1 dropout)

Output layer (4 classes)

Clone this project to train this model

Model

Model version: ⓘ Quantized (int8)

Last training performance (validation set)

ACCURACY 72.1%

LOSS 0.65

Confusion matrix (validation set)

	GLASS	METAL	PAPER	PLASTIC
GLASS	58.5%	19.5%	3.7%	18.3%
METAL	8.2%	76.7%	1.4%	13.7%
PAPER	1.3%	0%	89.7%	9.0%
PLASTIC	20.9%	7.0%	7.0%	65.1%
F1 SCORE	0.62	0.76	0.89	0.64

Metrics (validation set)

METRIC	VALUE
Area under ROC Curve ⓘ	0.92
Weighted average Precision ⓘ	0.72
Weighted average Recall ⓘ	0.72
Weighted average F1 score ⓘ	0.72

Data explorer (full training set) ⓘ

On-device performance ⓘ

Engine: ⓘ EON™ Compiler (RAM optimized)

INFERENCING TIME 36 ms.

PEAK RAM USAGE 214.6K

FLASH USAGE 535.8K

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-mem ory errors on the microcontroller. If either file is missing, the script raises a cle ar exception message so the user knows to copy both the model and label file s 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 t o the serial console. This gives a “raw” view of the model output and the act ual speed in frames per second on the OpenMV H7. The script also finds the b est label for this frame by taking the index of the maximum score (scores.inde x(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 t wo lists, vote_labels and vote_scores, that store the best label and its confiden ce 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 vali dation set. With the MobileNetV2 96×96 0.35 backbone and data augmentati on 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 r eflect light strongly.
- Metal is in the middle of the two groups. It has decent precision and rec all, but it still gets confused with other shiny things.

Edge Impulse also gives the OpenMV H7 an estimate of how well it will work o n 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 show that the model fits well within the board's memory budg et and is fast enough to support several frames per second, even with the extr a work that Python has to do.

7.2 On-Device Qualitative Tests

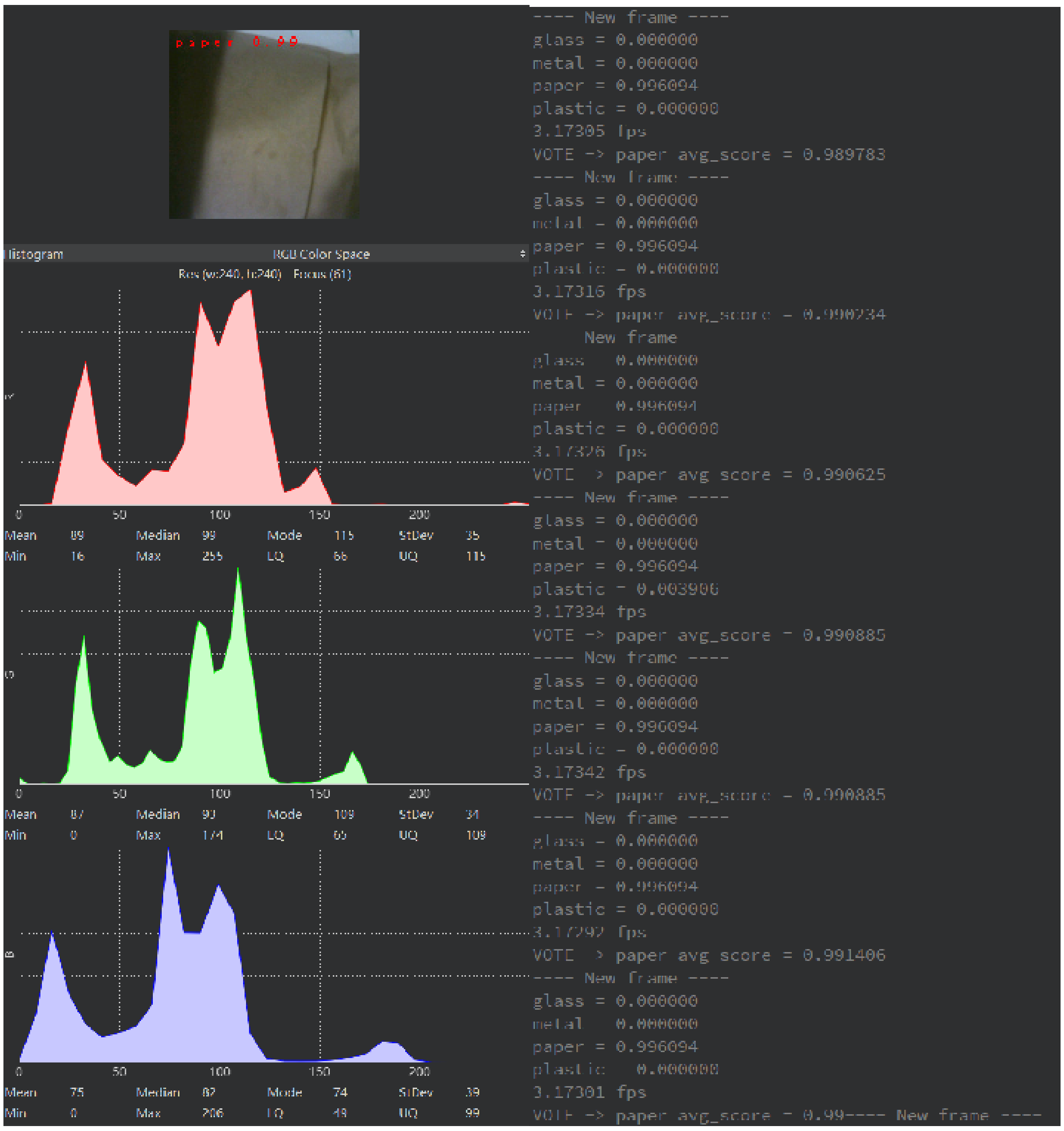
After that, we put the model on the OpenMV H7 and run live tests. The full pi peline (camera capture, preprocessing, TFLite inference, majority voting, and o verlay) runs on the device at about 3.17 FPS for all tests.

Case 1: Paper

We put a folded paper tissue in front of the camera and turned on the lights i nside. The final output is:

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

The predictions lock onto the right class after just a few frames and stay very s table. This is what we expected, since paper is one of the easiest categories.


paper 0.99
Histogram
RGB Color Space
Res (w:240, h:240) Focus (61)
0 50 100 150 200
Mean 89 Median 99 Mode 115 StDev 35
Min 16 Max 255 LQ 66 UQ 115
Mean 87 Median 93 Mode 109 StDev 34
Min 0 Max 174 LQ 65 UQ 109
Mean 75 Median 82 Mode 74 StDev 39
Min 0 Max 206 LQ 48 UQ 99

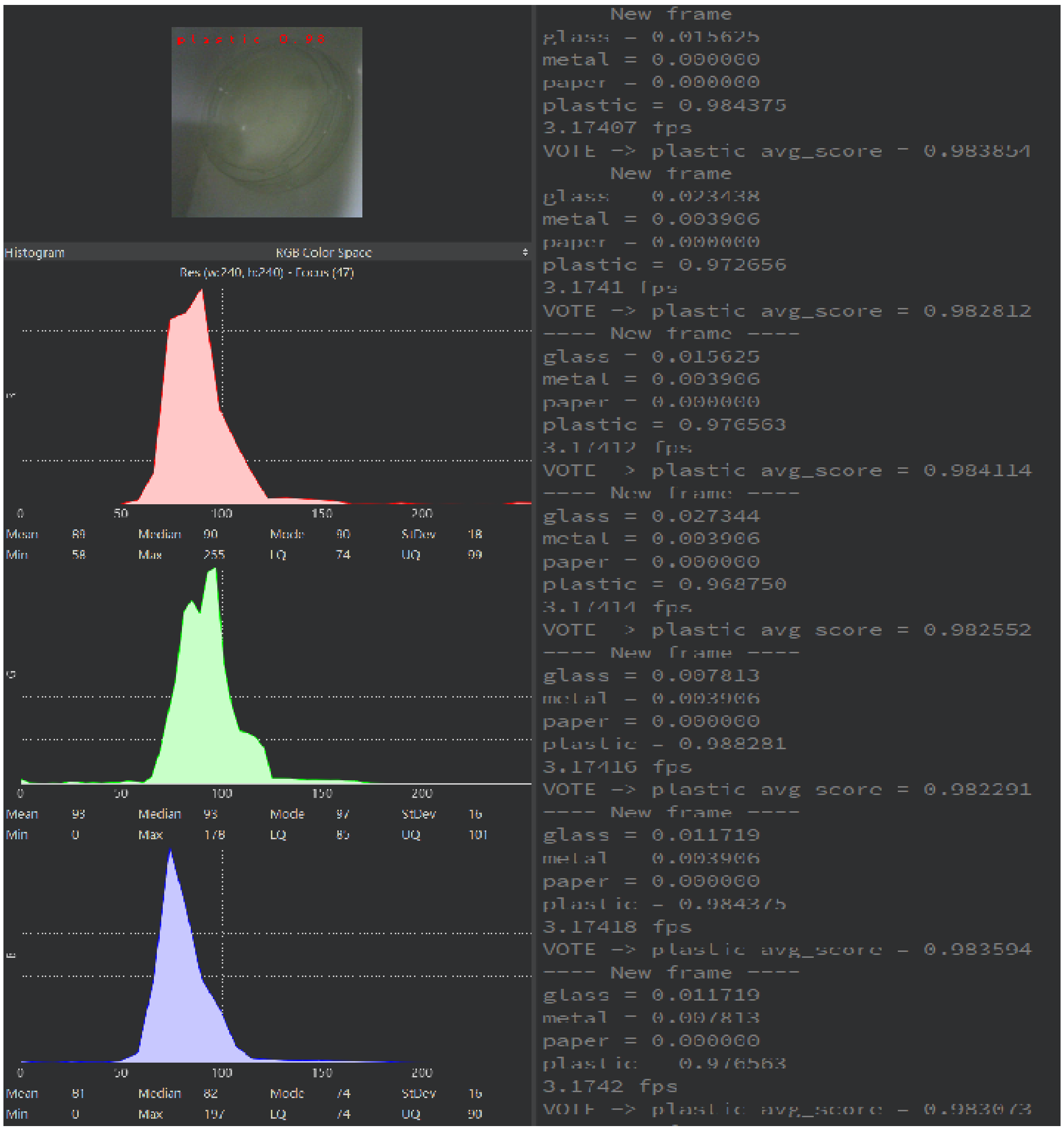
```
---- New frame ----
glass = 0.000000
metal = 0.000000
paper = 0.996094
plastic = 0.000000
3.17305 fps
VOTE -> paper avg_score = 0.989783
---- New frame ----
glass = 0.000000
metal = 0.000000
paper = 0.996094
plastic = 0.000000
3.17316 fps
VOTE -> paper avg_score = 0.990234
---- New frame ----
glass = 0.000000
metal = 0.000000
paper = 0.996094
plastic = 0.000000
3.17326 fps
VOTE -> paper avg_score = 0.990625
---- New frame ----
glass = 0.000000
metal = 0.000000
paper = 0.996094
plastic = 0.003906
3.17334 fps
VOTE -> paper avg_score = 0.990885
---- New frame ----
glass = 0.000000
metal = 0.000000
paper = 0.996094
plastic = 0.000000
3.17342 fps
VOTE -> paper avg_score = 0.990885
---- New frame ----
glass = 0.000000
metal = 0.000000
paper = 0.996094
plastic = 0.000000
3.17292 fps
VOTE -> paper avg_score = 0.991406
---- New frame ----
glass = 0.000000
metal = 0.000000
paper = 0.996094
plastic = 0.000000
3.17301 fps
VOTE -> paper avg_score = 0.99---- New frame ----
```

Case 2: Plastic

Next, we try it out with a plastic lid. The device again makes a clear majority:

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

The probabilities for plastic are the most important in almost every frame, and the majority vote just smooths out small changes. This shows that you can reli ably identify common plastic items in real time.


plastic 0.98
Histogram
RGB Color Space
Res (w:240, h:240) - Focus (17)
Mean 88 Median 90 Mode 90 StDev 18
Min 58 Max 255 LQ 74 UQ 99
Mean 93 Median 93 Mode 97 StDev 16
Min 0 Max 178 LQ 85 UQ 101
Mean 81 Median 82 Mode 74 StDev 16
Min 0 Max 197 LQ 74 UQ 90

```
     New frame
glass = 0.015625
metal = 0.000000
paper = 0.000000
plastic = 0.984375
3.17407 fps
VOTE -> plastic avg_score = 0.983854
     New frame
glass   0.023438
metal = 0.003906
paper = 0.000000
plastic = 0.972656
3.1741 fps
VOTE -> plastic avg_score = 0.982812
---- New frame ----
glass = 0.015625
metal = 0.003906
paper = 0.000000
plastic = 0.976563
3.17412 fps
VOTE  > plastic avg_score = 0.984114
---- New frame ----
glass = 0.027344
metal = 0.003906
paper = 0.000000
plastic = 0.968750
3.17414 fps
VOTE  > plastic avg score = 0.982552
---- New frame ----
glass = 0.007813
metal = 0.003906
paper = 0.000000
plastic = 0.988281
3.17416 fps
VOTE -> plastic avg score = 0.982291
---- New frame ----
glass = 0.011719
metal   0.003906
paper = 0.000000
plastic = 0.984375
3.17418 fps
VOTE -> plastic avg_score = 0.983594
---- New frame ----
glass = 0.011719
metal = 0.007813
paper = 0.000000
plastic   0.976563
3.1742 fps
VOTE -> plastic avg_score = 0.983073
```

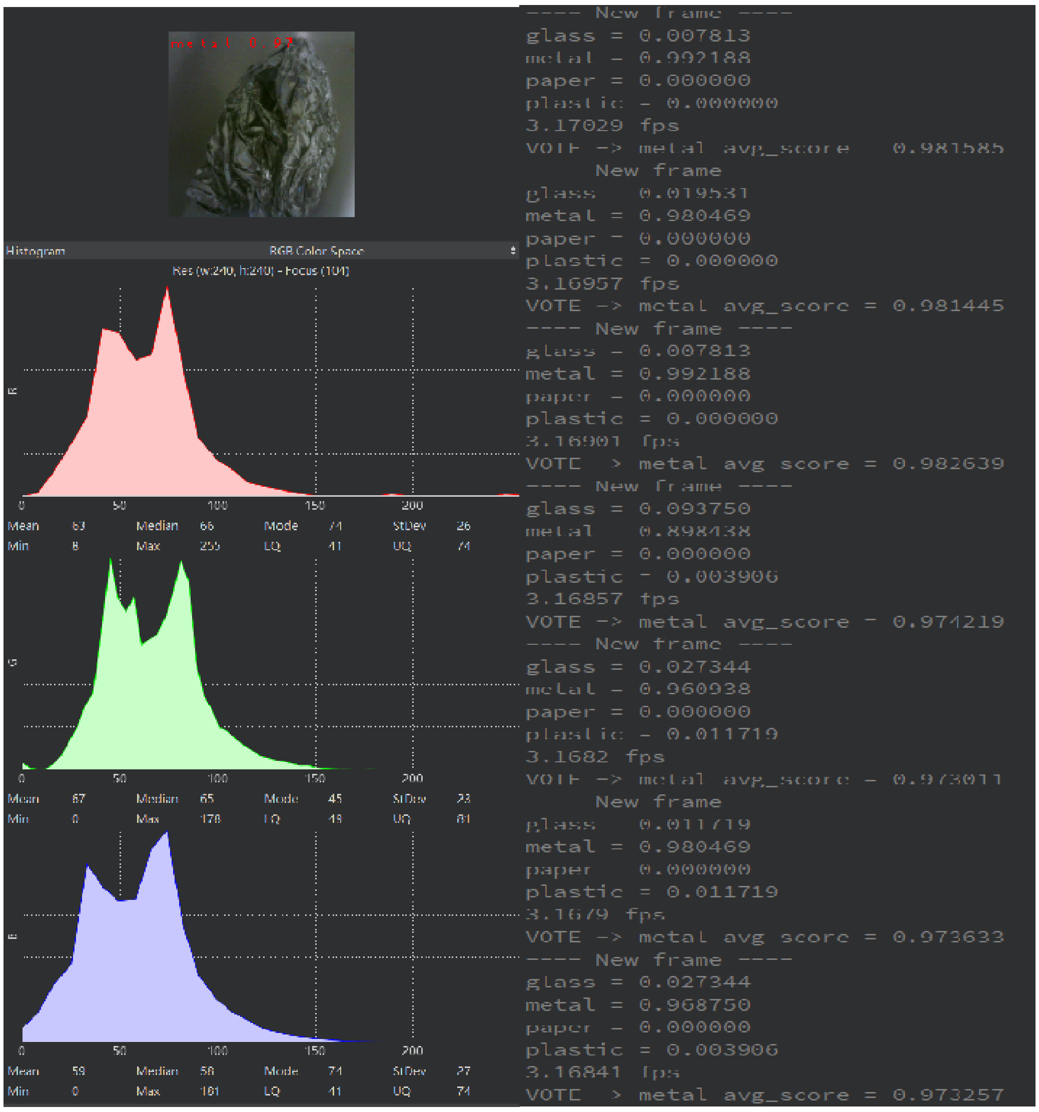
Case 3: Metal

We use a crumpled aluminum foil ball for the metal class. This makes it harder because the reflections and specular highlights are so strong.

- Last label: metal

- Confidence level: about 0.97
- Speed: 3.17 frames per second

The per-frame scores do change more than in the paper and plastic cases, but the voted label is still correct and fairly stable. This shows that the model has l earned useful features for metal, even though it is not always the same.


metal 0.97
Histogram
RGB Color Space
Res (w:240, h:240) - Focus (101)
Mean 63 Median 66 Mode 74 StDev 26
Min 8 Max 255 LQ 41 UQ 74
Mean 67 Median 65 Mode 45 StDev 23
Min 0 Max 178 LQ 48 UQ 81
Mean 59 Median 58 Mode 74 StDev 27
Min 0 Max 181 LQ 41 UQ 74

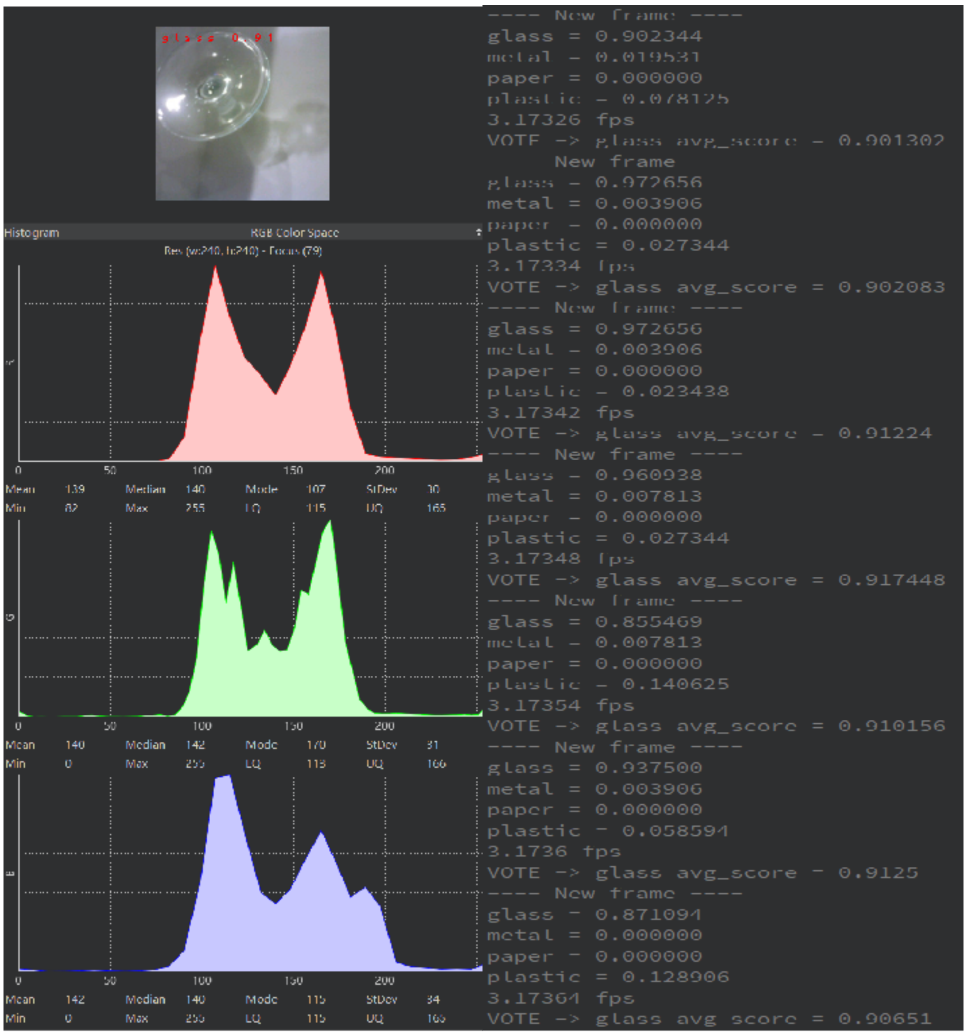
```
---- New frame ----
glass = 0.007813
metal = 0.992188
paper = 0.000000
plastic = 0.000000
3.17029 fps
VOTE -> metal avg_score = 0.981585
---- New frame ----
glass = 0.019531
metal = 0.980469
paper = 0.000000
plastic = 0.000000
3.16957 fps
VOTE -> metal avg_score = 0.981445
---- New frame ----
glass = 0.007813
metal = 0.992188
paper = 0.000000
plastic = 0.000000
3.16901 fps
VOTE -> metal avg_score = 0.982639
---- New frame ----
glass = 0.093750
metal = 0.898438
paper = 0.000000
plastic = 0.003906
3.16857 fps
VOTE -> metal avg_score = 0.974219
---- New frame ----
glass = 0.027344
metal = 0.960938
paper = 0.000000
plastic = 0.011719
3.1682 fps
VOTE -> metal avg_score = 0.973011
---- New frame ----
glass = 0.011719
metal = 0.980469
paper = 0.000000
plastic = 0.011719
3.1679 fps
VOTE -> metal avg_score = 0.973633
---- New frame ----
glass = 0.027344
metal = 0.968750
paper = 0.000000
plastic = 0.003906
3.16841 fps
VOTE -> metal avg_score = 0.973257
```

Case 4: Glass

Lastly, we test a glass cup that has reflections, transparency, and an effect on t he background.

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

Here, the raw probabilities change a lot when the cup or the lighting changes. However, the sliding-window vote helps keep the prediction stable and still gives the right label most of the time.


glass 0.91
Histogram
RGB Color Space
Res (w:240, h:240) - Focus (79)
Mean 139 Median 140 Mode 107 StDev 30
Min 82 Max 255 LQ 115 UQ 165
Mean 140 Median 142 Mode 170 StDev 31
Min 0 Max 255 LQ 113 UQ 166
Mean 142 Median 140 Mode 115 StDev 34
Min 0 Max 255 LQ 115 UQ 165

```
---- New frame ----
glass = 0.902344
metal - 0.019531
paper = 0.000000
plastic - 0.078125
3.17326 fps
VOTE -> glass avg_score - 0.901302
     New frame
glass - 0.972656
metal = 0.003906
paper - 0.000000
plastic = 0.027344
3.17334 fps
VOTE -> glass avg_score = 0.902083
---- New frame ----
glass = 0.972656
metal - 0.003906
paper = 0.000000
plastic - 0.023438
3.17342 fps
VOTE -> glass avg_score - 0.91224
---- New frame ----
glass - 0.960938
metal = 0.007813
paper - 0.000000
plastic = 0.027344
3.17348 fps
VOTE -> glass avg_score = 0.917448
---- New frame ----
glass = 0.855469
metal - 0.007813
paper = 0.000000
plastic - 0.140625
3.17354 fps
VOTE -> glass avg_score = 0.910156
---- New frame ----
glass = 0.937500
metal = 0.003906
paper = 0.000000
plastic - 0.058594
3.1736 fps
VOTE -> glass avg_score - 0.9125
---- New frame ----
glass - 0.871094
metal = 0.000000
paper - 0.000000
plastic = 0.128906
3.17364 fps
VOTE -> glass avg score = 0.90651
```

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 th e OpenMV H7. But the tests also show a clear pattern:

- The most stable and accurate classes are usually paper and plastic. It's e asier to tell them apart from the background, so the model converges q uickly 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 te sts.

- 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 pre processing, and the sliding-window voting scheme gives useful predictions in a lot of real-world situations. It also sets a good standard for future improvem ents in data collection and model tuning on MCU hardware.

8. Limitations

Our current prototype still has several important limitations:

1) The dataset and deployment are in different domains.

The model is trained on TrashNet images taken with smartphones in certain li ghting and backgrounds. However, on the OpenMV H7, we see different view s, resolutions, and settings. When we go from clean validation images to real s cenes in front of the camera, this mismatch makes the system less reliable.

2) Classes have different levels of difficulty.

People are usually very sure about predicting paper and plastic, but metal and glass are still very hard to predict. Their looks change a lot depending on refle ctions, transparency, and background colors. This makes the chances change more, and mistakes happen more often.

3) Real-time and resource limits that are very tight.

The full pipeline on the OpenMV H7, which includes capturing images, prepro cessing them, running TFLite inference, voting, and drawing on the screen, run s at about 3.17 frames per second. This speed is fine for a demo, but it doesn't leave much room for more processing or a more complicated model.

4) Limited coverage for evaluations.

We only use a small number of household items and indoor scenes for our on -device tests. We don't have a big, organized test set that includes a lot of diff erent backgrounds, lighting conditions, and camera angles yet.

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 bac kgrounds, distances, and lighting setups. Training (or fine-tuning) on this data from the same domain should close the gap between domains and make it m ore 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 designin g targeted data augmentation for reflective and transparent objects (for exam ple, by making brightness and contrast changes stronger, adding blur, or addi ng background noise). You could also try class-balanced sampling or loss func tions.

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 eas ier, 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 mo re aggressive quantization schemes. Try to keep the accuracy as close to the c urrent baseline as possible. Increase testing, and the explanations are easier to understand for people who aren't experts.

5) Increase testing and the experience of users.

Test the system in more realistic settings, like labs, shared kitchens, or fake rec ycling stations. Also, improve the user interface (LEDs, sounds, or icons) so tha t predictions are easier to understand for people who aren't experts.

10. Conclusion

We made a complete edge-AI waste classification system for this project that works only on the OpenMV H7 microcontroller board. We made a TinyML pip eline using the TrashNet dataset and the Edge Impulse platform. It includes im age preprocessing, transfer learning with MobileNetV2 96×96 0.35, and deplo yment as a TensorFlow Lite model that works with OpenMV. The classifier gets about 72.1% validation accuracy on four classes (glass, metal, paper, and plasti c) and fits within the device's memory limits (≈215 KB RAM, ≈536 KB flash). It also works in real time at about 3.17 FPS on the camera stream.

Our Python script on the device combines camera capture, model inference, a sliding-window majority vote, and on-screen overlays to make stable, human-readable predictions for everyday things like cups, lids, foil balls, and paper tis sues. The prototype clearly shows that useful waste-image recognition is possi ble on cheap, low-power hardware, even though the system has only been tes ted in a few places, and accuracy on metal and glass is still limited. This work e stablishes a clear foundation for subsequent enhancements in data collection, model architecture, and MCU-level optimization, demonstrating how design d ecisions at each phase of the pipeline directly influence performance and user experience in resource-limited AI systems.

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