AUTONOMOUS SYSTEM FOR SORTING OBJECTS AT THE EDGE

Geffen Cooper, Vincent Benenati, Bethany Long, Kat Copeland,
Tyler Ekaireb, Satish Kumar, B.S. Manjunath, and Yogananda Isukapalli
University of California, Santa Barbara
Santa Barbara, CA 93106
{geffen_cooper, vincentbenenati, bethanylong, kcopeland,
tylerekaireb, satishkumar, manj, yoga}@ucsb.edu

ABSTRACT

This paper describes an implementation of an end-to-end machine learning-based system for sorting objects. We explore the specific application of sorting recyclables at the edge. This stands in contrast from existing waste processing systems that aim to classify and sort a large variety of items coming from multiple waste streams. Moving the sorting process closer to the point of waste generation reduces the risk of contaminating recyclables and enables local data collection to train the classification system on specific waste sources. This eases the classification task which enables the use of cheap, low-power electronics. Our system uses a low-power microcontroller (MCU) with an on-board camera module and a convolutional neural network (CNN) accelerator for classifying items. The MCU also controls a set of mechanical arms to physically sort objects that move along a conveyor belt. We outline the specifications of the physical system in addition to the development process of our machine learning model.

INTRODUCTION

The ability to autonomously sort objects is important for contexts such as quality assurance and waste management as it can potentially increase productivity and reliability at a lower operating cost than manual sorting. However, automated sorting systems are mainly limited to centralized factory environments as the sorting process is typically complicated and requires industrial grade mechatronic systems which are expensive and high power. In this paper we describe a decentralized approach in which the sorting task is moved to *the edge* [1] where items are sourced. The source location has a reduced number of object types and operates at a smaller scale which enables the use of low-power and low-cost systems rather than highly complex centralized systems.

One of the main challenges of automating the sorting process is the complexity of object identification. Approaching this as an image-based classification task is favorable due to wide support for vision-based systems. Machine learning-based approaches have been very successful for image classification and detection tasks. Despite their success, they are computationally intensive and typically require high compute resources. However, in recent years there has been immense

progress in hardware and software for machine learning [2] to make image classification and detection on an edge device feasible. In the proposed end-to-end method, we leverage a new MCU (MAX78000) with a CNN accelerator optimized for near real-time image classification.

Our system classifies and sorts translucent plastic, metal cans, and crushed paper. This involves doing real-time image classification followed by a mechatronic system for physically sorting the items that move along a conveyor belt. This system can be specialized to its source location such as schools, restaurants, and offices to sort bottles, cans, and paper. This is easier than trying to sort multiple waste streams from different sources simultaneously which introduces contamination and merges multiple distributions of data. In addition, edge systems can collect and save data about their source distribution for further training. The rest of the paper describes the design process and constraints of our sorting system with each subsystem described in detail.

PHYSICAL SYSTEM DESIGN

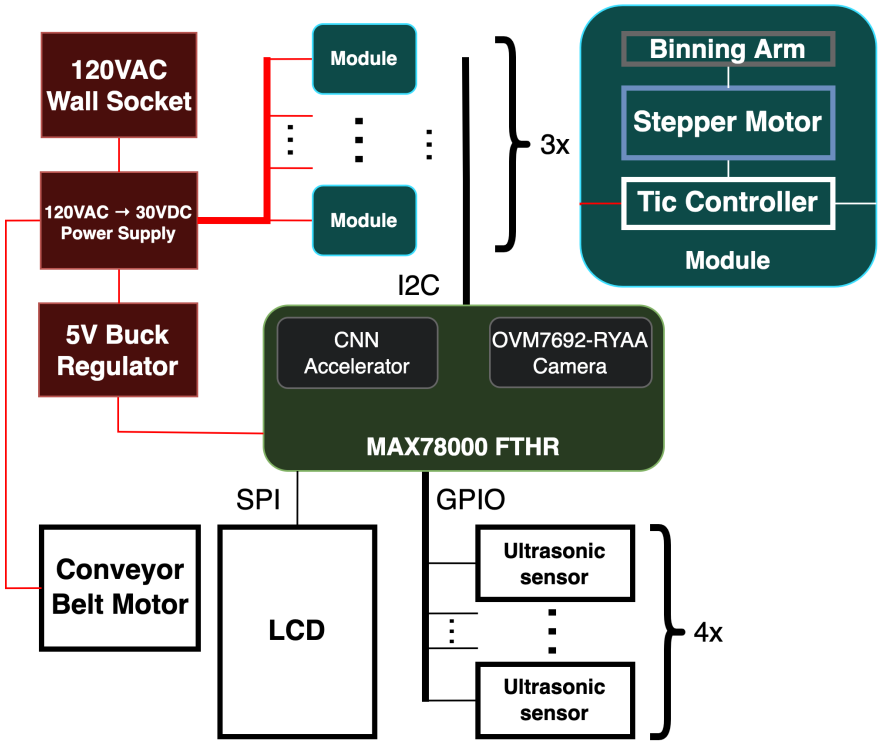
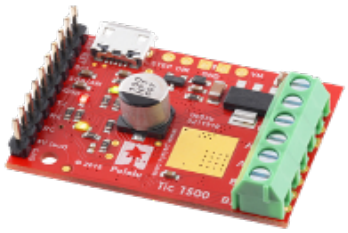
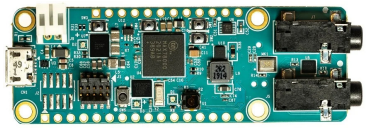


Figure 1: Full System Block Diagram



(a) Tic Stepper Motor Controller



(b) MAX78000FTHR Development Board

Figure 2

Our physical system consists of five main components: the MAX78000FTHR development board, ultrasonic sensors, stepper motors, conveyor belt, and power supply. Accented around the system is 80/20 mounting, 3D printed components and limit switches to regulate movement of the lever arms. We were able to successfully integrate and orchestrate a system with multiple protocols such as SPI, I2C, and GPIO, which all play an integral part in the full system.

MECHANICAL SETUP

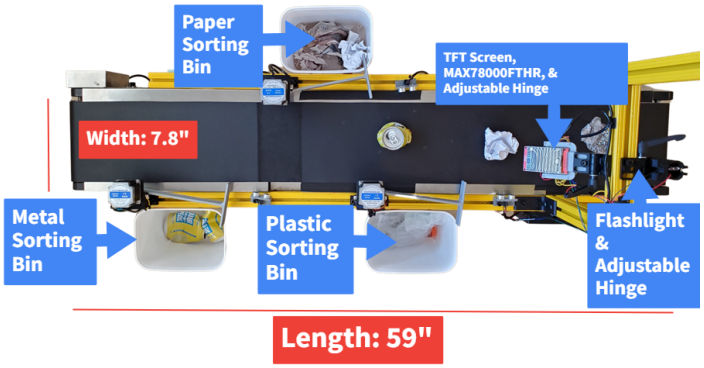


Figure 3: Top System View

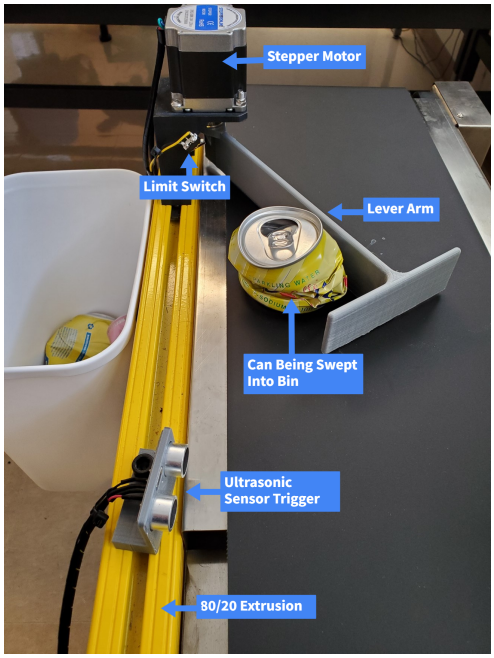


Figure 4: Arm Assembly

The mechanical construction of this system was largely constrained by the available tooling, time, and technical expertise. A preassembled conveyor belt was purchased that could translate items and provided ample space for many handheld consumable items, e.g. soda cans and plastic bottles. To allow for rapid prototyping, 80/20 extrusion was used to frame the conveyor belt in order to provide a sturdy and easily adjustable mounting platform for the hardware components. Locking hinges were used to provide a secure method of testing different camera and flashlight angles and to keep the angles constant once desirable ones were found as seen in Figures 3. While pneumatic actuators would be desirable to push objects off of the conveyor belt with a linear motion, the noise of the associated compressor made them prohibitive. Stepper motors with 3D printed lever arms attached with shaft hubs were chosen instead for their mechanical and control simplicity, as seen in Figure 4. The motors were specified in order to provide more than sufficient acceleration and actuating force even at the tip of the lever arm. Custom 3D printed parts were made to mount the stepper motors with their limit switches, the flashlight, the MCU, the power-supply unit, and ultrasonic sensors to the 80/20 frame.

ELECTRONIC HARDWARE COMPONENTS

MAX78000FTHR: At the center of it all, the MAX78000FTHR board shown in Figure 2B is doing the bulk of computation and management of the full system. We chose the MAX78000 board because it has an onboard camera module integrated with a convolutional neural network (CNN) hardware accelerator, which makes real time computer-vision classification possible despite the

board's small form factor and low power capabilities. The board also has enough GPIO, I2C, and SPI pins to interface with all of our peripherals.

Stepper Motors: The NEMA 23 bipolar stepper motors shown in Figure 4 had 1.26Nm of holding torque that allowed us the versatility of making precise motor movements with variable torque and speed. The faster motor profile during the "blocking phase" allowed the arm to extend extremely quickly at the cost of torque and the "pulling phase" profile allowed the arm to retract, pulling heavy objects into their associated bins with higher torque but with half the speed. Limit switches were used to calibrate the motors range as well as to home them periodically.

Tic Controllers: The Pololu Tic T500 Stepper Motor Controllers greatly simplified the process of configuring and controlling the stepper motors. The device also has an extensive user guide which made it highly accessible. The MAX78000 communicates with the Tics via an I2C connection. We chose I2C because it is easily expandable to more peripheral devices while only requiring 2 wires due to its addressable format. However, if the system were to be expanded significantly, it would be inadvisable to continue to use I2C because the protocol struggles with long distance communication without a I2C repeater.

Ultrasonic Sensors: We placed the HC-SR04 ultrasonic sensors in 4 key locations to keep track of objects as they travelled along the conveyor belt. When an object passes an ultrasonic sensor, it triggers an interrupt which then initiates an action depending on where on the conveyor belt the sensor is located.

Power Supply: The system can be powered from a single outlet. The conveyor is powered by a 120VAC outlet that is also fed into a switching power supply that outputs 30VDC for the stepper motors. A buck converter downshifts the 30VDC to 5VDC for the MCU, Tic controllers, flashlight, and ultrasonic sensors.

SYSTEM CONTROL FLOW

The sorting system is largely interrupt-driven, with ultrasonic distance sensors triggering most activity. The sensors sit beside the camera's field of view and each bin. They are used to trigger the camera for inference and update the system queuing system, which maintains the location of each item on the conveyor belt.

OVERVIEW

Sorting process: Consider the diagram in Figure 5. An ultrasonic distance sensor monitors the region of the conveyor belt below the camera. Once an object enters the camera's field of view, the ultrasonic distance sensor will be activated and trigger an interrupt (1) to capture an image and begin inference (2). Once the inference process is complete, the CNN will have produced an expected category to which the object belongs, e.g., metal. An event is then added to the queues (3) corresponding with each bin that the object is expected to pass on its journey across the conveyor

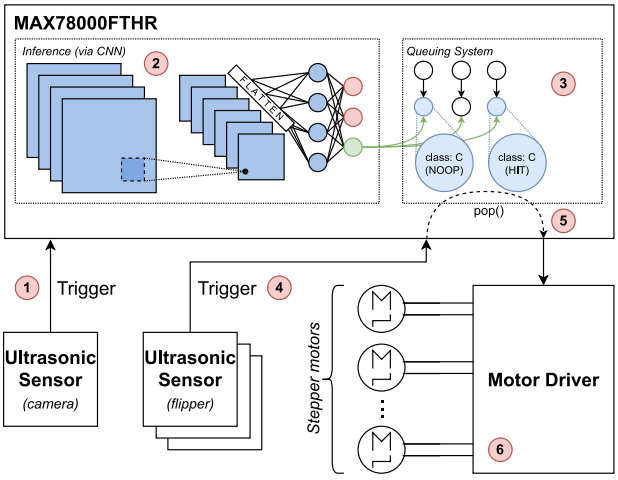


Figure 5: Overview of system control flow

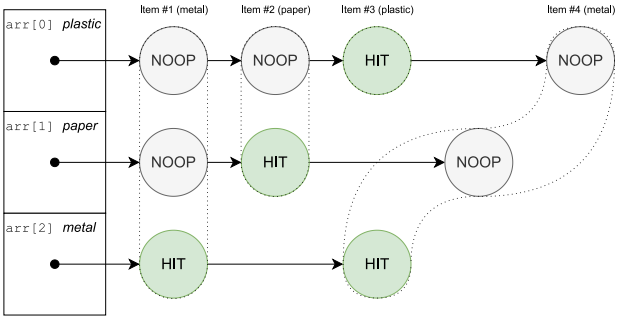


Figure 6: State of the queues immediately after identifying a metal object, then a paper object, then a plastic object, then another metal object

belt. Assume that the object is destined for the third bin; the queues associated with the first and second bins will each receive a `NOOP` event, which will ensure that they do nothing as the object passes them. The third queue, whose associated bin is the intended destination, will receive a `HIT` event. Each bin is outfitted with a dedicated ultrasonic distance sensor. When the sensor detects an item in front of its bin (4), it triggers the system to pop the next event from the queue (5). If this event is a `NOOP`, then nothing will be done. If the event is a `HIT`, then motor driver will be signalled to activate the lever arm (or "flipper") and push the object into the bin (6).

TASK SCHEDULING

Queuing system: At the heart of the system lies a software queuing class, `Sorter`, that maintains the approximate location of items on the conveyor belt and the categories to which they belong. `Sorter` holds one queue for each item category, where each node represents an ultrasonic sensor detection event that will occur in the future. If the item associated with a particular node is of the same category as the bin associated with the queue, then the node will be marked as a `HIT` event; otherwise, the node will be marked as a `NOOP` event. An example of the queues in a state after several detection events is shown in Figure 6. `Sorter` also provides several functions to interact with the struct more easily, as shown in Figure 7. The `Sorter` class design provides several benefits. Because all events are held in queues with head and tail pointers, all push and pop operations are performed in constant time, allowing the system to operate smoothly at high rates of speed and meet timing requirements while executing on a low-power MCU. The speed of the provided functions makes them well suited to invocation from the ultrasonic sensor interrupt handlers. The design can also be trivially extended to arbitrary numbers of categories and items; in fact, the initialization function accepts both of these values as parameters upon instantiation.

Timing Constraints and Scheduling: When the conveyor belt is at full speed, there is only one second between the time an image of an object is captured and the time it reaches the first bin. The time between an object passing an ultrasonic sensor and missing the window for a lever arm to swing out is even more slim. Therefore, time is of the essence in all aspects of system, but

```
// Add detected item based on CNN's predicted class
void sorter__add_item(sorter *s, int item_type);

// Update queues when a bin's ultrasonic sensor detects an object
// Returns: 1 if flipper should activate (HIT), else 0 (NOOP)
bool sorter__detected_item(sorter *s, int flipper);

// Initialize Sorter with an arbitrary number of categories
sorter sorter__init(int num_types, int queue_size);
```

Figure 7: Some of the available functions to interact with Sorter

specifically when it comes to handling ultrasonic sensor interrupts, classifying objects, updating the sorting queues, and sending commands to the stepper motors via the Tic controllers.

Ultrasonic sensor interrupts can occur in rapid succession so their interrupt service routines (ISR) cannot be computationally heavy. Currently, there are no tools provided by Analog Devices for integrating an RTOS on the MAX78000FTHR so we implemented a simple scheduling system for our tasks. We opted for a flag-polling procedure in the grand loop of our code, so that an ultrasonic sensor's ISR only needs set its associated flag to 1. Then all flags are checked sequentially in the grand loop and can initiate camera captures, classification, and motor movement.

The inference latency of our classification model is consistently 13.5ms. Subsequently updating the sorting queues also takes a matter of milliseconds, so despite there being less than 1 second between camera capture and the first category's ultrasonic sensor, our system consistently makes the deadline.

Communicating with the Tic controllers via I2C, however, was much less reliable. The latency of a single, successful command via I2C is well within the timing constraints set by the speed of the conveyor belt, with a max translation of 0.5m/s. Unfortunately, the I2C in our system was prone to interference due to the proximity of components. We mitigated this interference to an extent by implementing timeouts and reinitialization in our I2C protocol if an error should occur. In future work we would make our system more robust by using shielded cables and enclosures to minimize interference, and experiment with utilizing a different protocol such as UART. Currently, successive I2C failures can result in a "miss" by the lever arms due to the accumulated latency.

OBJECT IDENTIFICATION

The object identification task involves real-time image classification of moving items on the conveyor belt. We have four classes: `Plastic`, `Paper`, `Metal` and `None`. These categories are different enough in their features to make classification feasible but instances within a class are variable enough in color, size, and shape to make the task realistic and challenging. The following subsections describe the strategies employed to build a robust model that overcomes the challenges

of working with live data from the onboard camera.

DATA COLLECTION

The challenge in working with live data rather than a fixed dataset is ensuring that the model is robust to the variability of the input that will be seen at inference time. A common assumption made in machine learning is that data seen during inference comes from the same distribution as the training data. Therefore, we created our own dataset under the testing time conditions of the model. For example, rather than collecting images using a phone camera or online dataset, we captured the images using the onboard camera while the objects were moving on the conveyor belt.

To motivate this further, table 1 compares images from different sources. Images from the onboard camera are low quality and have some distortion. During inference, the model must be able to classify these lower quality images. By adjusting the camera module's default settings (automatic gain ceiling, exposure time) we increased the quality of the images as shown in the 'default' and 'custom' columns of table 1. We collected approximately 4,000 labeled images for classification and further labeled 1,500 of these with bounding boxes for object localization. Due to resource and timing constraints we were limited in the amount of data we could collect.

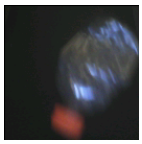
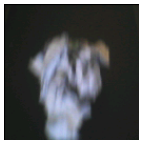
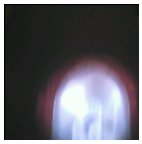
Class	Default	Custom	Phone Camera
Plastic			
Paper			
Metal			

Table 1: Comparison of Image Sources

MODEL ARCHITECTURE

While the MAX78000 makes running CNNs on an MCU feasible, it comes with hardware constraints that makes developing models challenging. In general, it is not feasible to directly run state of the art CNN models on the board due to memory and architecture limitations. These constraints include 432 KiB of memory for storing the model weights, 512 KiB of data memory for intermediate storage during a forward pass, quantization of the model parameters to 8-bit values, and limitations on the model architecture such as kernel size, input dimensions, and types of connections and layers. Therefore, we developed our own CNN architecture inspired by VGG [3]

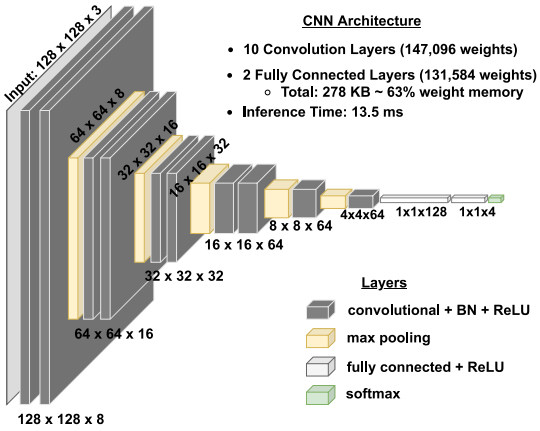


Figure 8: CNN Classification Architecture

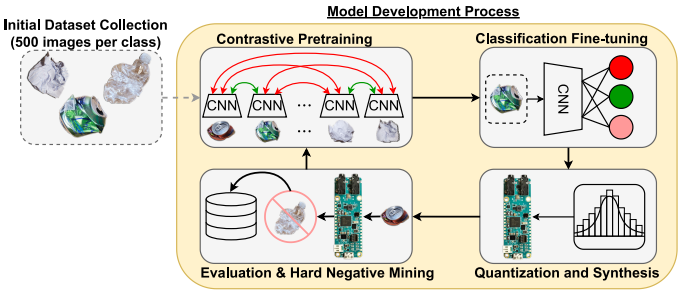


Figure 9: Iterative Training Procedure

due to its success and simplicity. The complete hardware specifications are discussed in [4] and [5].

Our architecture is shown in Figure 8. It emulates the structure of VGG in terms of layer type except that we add batch normalization. This architecture uses 278 KiB for the weights (63% of weight memory) and has an inference latency of 13.5 ms. For object localization, we extend the output layer to have four additional outputs for the coordinate of the top left corner and the width and height of the bounding box, (x, y, w, h).

TRAINING PROCEDURE

During preliminary testing of our classification model on live data, we found that it performed significantly worse than the test set accuracy. Therefore, we developed a training procedure to iteratively improve our model. This procedure is described below and summarized in figure 9.

1. Initial Dataset Collection: 500 images per class were collected as a starting point for training. These images have a dimension of 3x128x128 (CxWxH) with a pixel format of RGB565 (2 bytes per pixel). The images were collected indoors while the conveyor belt was moving with overhead lighting from a flashlight. We split our dataset 85% and 15% for training and testing respectively.

2. Contrastive Pretraining (CP): CP is a technique where the model ideally learns representations of the data such that semantically similar images will be close together and semantically different images will be far apart. For example, if we augment (e.g. rotate, shift the color) an image of a can the semantic information does not change; i.e., it is still identifiable as a can. In CP, we want representations of augmented cans to be close together, where closeness can be interpreted as the euclidean distance of the output embeddings produced by our CNN. In this way the model ideally learns which features are more important for distinguishing each class. Specifically we use the framework outlined in [6]. One of the benefits of CP is that it can be implemented in a self-supervised manner (without labeled data). If we deployed our sorting system, we would be able to collect images during normal operation and train our model to learn useful features without needing any labels.

3. Classification Fine-tuning: After pretraining, the model is fine-tuned for image classification which requires labeled data. However, [6] shows that CP can dramatically reduce the amount of labeled data required to fine-tune an image classification model. To get the best results we use all of the labeled data. We do CP for 200 epochs with a learning rate of 0.0005 and a batch size of 256 with the Adam optimizer. We then fine-tune all the layers for 100 epochs with a learning rate of 0.001 and a batch size of 64 with the Adam optimizer. The results are summarized in table 2.

Training Method	Labeled Training Data (% of total)	Classification Accuracy (%)
Fully Supervised	100	96.5
Fully Supervised	10	25
CP + Supervised Fine-tuning	10	80
CP + Supervised Fine-tuning	100	98.5

Table 2: Classification accuracy on test set under different training conditions

From table 2 we see that CP followed by fine-tuning on the full labeled training set results in the best accuracy. However, this method also takes the longest to train because CP requires many epochs with a large batch size (256) to ensure enough hard negative samples are seen. When we use only 10 % of the labeled data, CP has a substantial effect on the classification accuracy. Without CP, the model gets 25% accuracy which is equivalent to random guessing. Depending on the amount of labeled data CP can have limited benefits.

4. Quantization and Synthesis: All model development is done using the MAX78000 software tools which implement a modified version of PyTorch that augments the standard modules with information about the hardware (clipping, rounding, fusing layers) so that models can be synthesized onto the accelerator. They also employ quantization aware training which helps reduce the loss in accuracy when a model gets quantized to 8-bit values. This mapping of a trained model onto the MAX78000's CNN accelerator is referred to as the quantization and synthesis step in figure 9.

5. Evaluation and Hard Negative Mining: Once the model is loaded onto the board, it is evaluated on live data. We employ a strategy called hard-negative mining to find examples of images which the model misclassified. In software we fix the class we want to test. If the model predicts a different class, then we add the image to our dataset. In this way, we find the model's weak points and give it the most informative examples to train on. With new hard-negative samples collected, the process is repeated starting from the contrastive pretraining stage.

FUTURE WORK

Currently we do not have measurements estimating the accuracy or speed of the system during runtime. Given more time we would measure how many items out of a batch (e.g. of size 100) were correctly sorted and the amount of time required to go through the batch. Potential improvements to the system include designing a more compact mechanical sorting mechanism, expanding the number of object classes, integrating an external camera module, increasing the amount of training

data, and experimenting with more complex architectures for multi-object recognition.

CONCLUSIONS

Overall the system was able to successfully sort objects under consistent lighting conditions and with sufficient spacing between each item. A demo of the full system can be seen at [7]. Our end-to-end implementation shows that it is feasible to sort objects on an edge device in real-time with limited space and components. Finally, we showed that the MAX78000 is capable of object localization in addition to classification.

ACKNOWLEDGMENTS

We would like to thank Brian Rush and Robert Muchsel from Maxim Integrated for their technical assistance in working with the MAX78000.

REFERENCES

[1] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[2] T. Sipola, J. Alatalo, T. Kokkonen, and M. Rantonen, "Artificial intelligence in the iot era: A review of edge ai hardware and software," in *2022 31st Conference of Open Innovations Association (FRUCT)*, pp. 320–331, 2022.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[4] "Max78000 artificial intelligence microcontroller with ultra-low-power convolutional neural network accelerator - maxim integrated." `https://www.maximintegrated.com/en/products/microcontrollers/MAX78000.html`.

[5] "Maxim integrated ai github repository." `https://github.com/MaximIntegratedAI`.

[6] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," 2020.

[7] "Scrap sort demo video." `https://www.youtube.com/watch?v=CiYKWzBC99E`.