TRASH.PY - A SMART SYSTEM ENSURING PROPER WASTE SORTING AND ELIMINATING USER DECISIONS

Ethan Davis and Marcela Lebrija
Academic Advisor: Prof. Steven Skerlos

INTRODUCTION

For proper waste processing, the trash cans at the University of Michigan need to separate waste into different categories. These categories include landfill and recycling, and sometimes some others. The current burden for separating this waste is placed on the trash can user, with different labeled receptacles being provided for the user to decipher and manually separate their waste between. Relying on users to properly classify their waste causes problems. For instance, some users do not care enough to throw their waste into the proper receptacles, resulting in the recyclables becoming contaminated, and potentially recyclable items being taken to a landfill. Additionally, one of the receptacles could become overfull, resulting in the users just throwing all their waste into the only available receptacles. The recycling system at the university also differs from that in Ann Arbor and leads to confusion. Since most students and faculty live off campus, they are used to doing their household recycling for the Ann Arbor standard. We aim to address these problems by redesigning the trash cans at the University with an autonomous waste separation system. Removing the decision from the user on whether their item is recycling or landfill and therefore reducing the contamination in the bins. The bin will be able to detect whether an item dropped into the bin by a user is meant for landfill or recycling, and then navigate that item to the proper storage receptacle contained within the bin. This system will be based on a computer vision model that will be trained to understand the proper sorting standards at the university. We aim to have a trash can that yields more valid/clean recyclables than the current system at the University.

Contaminated recycling in public bins is a problem. At the university, recycling is collected and then sent to the Washtenaw County Materials Recovery Facility (MRF). There, items in the recycling stream are sorted through both by machine and by hand. These sorted items are then bundled together and sold off to different specialized recycling centers to then be reused into new products. If there are contaminants in the recycling stream, the MRF is where they are dealt with. Contaminants cause two main issues at the MRF. Firstly, the items that cannot be recycled clog up the system, taking additional time and energy to remove. The removed contaminants then have to be dealt with by shipping to a landfill. Secondly, contaminants can contaminate actual recyclable items, and reduce the total number of items that are actually recycled. An example of this would be liquids and other food items getting on paper products. The University is looking at changing to the Ann Arbor MRF in the next coming years. This will cause a shift in the recycling standards at the university since the MRFs have different rules and regulations.

Over the course of semester WN2022, we designed the trash.py bin prototype. It has a latch system to allow the intake of items, a staging area to hold the items for classification, and two internal bins for disposing of trash and recyclable waste, respectively. We built a physical prototype of the staging area and camera system –which works with a computer vision model and web server– to simulate a working prototype of a trash.py bin. In this simulation the web browser facilitates mechanical triggers in the bin, and the results from the model are displayed in the browser to simulate the bin sorting an item.

METHODS

Stakeholders

In order to start understanding the recycling process at Michigan we met with program managers at the Office of Campus Sustainability (OCS) in order to get more insight on the problem. OCS has a zero-waste initiative that is focused on cutting back the amount of waste getting sent to landfills by the university. They are looking at cutting back the amount of waste by 40% by 2025. Through this initiative they have gathered a lot of data on what are the root causes of bad recycling. From the data collected we were able to conclude that 40% of the recycling on campus is contaminated [1]. This contamination comes from dirty recyclables along with items that should not go into the recycling in the first place and can be broken down in figure 1. The University of Michigan's recycling program is different from the city of Ann Arbor and therefore causes confusion. The university is planning on moving towards sending their recyclables to the Ann Arbor MRF that deals with separating the recycling and getting it sent back for reuse. But for now, the university sends their recycling to a different MRF that has distinct rules for recycling such as no glass. Most university students don't live on campus and therefore reside in Ann Arbor, making their daily recycling at home different from when using University bins. For that reason, having an autonomous recycling system that can be updated from just retraining a model is a good solution for the vast amounts of different recycling systems to keep track of.

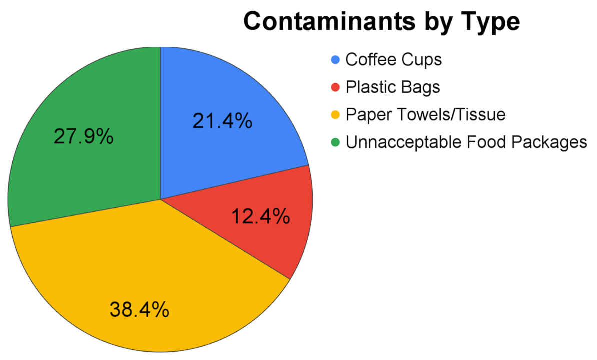


Figure 1. The different contaminants of recycling at the university by type. [1]

User Experience. When interacting with the trash.py bin there are two sets of main users. The front end being the students, faculty, and anyone else on campus that is throwing things into the bin. The back end user is the janitorial staff who services the bins and empties them out. Both of these groups are crucial to the success of the project.

In order to gain input from students, we developed a survey for the front end users in order to get some insights on their daily interactions with the current university bins. The anonymous survey featured 17 questions ranging from "How often do you use waste bins inside the university buildings to dispose waste in a day?" to yes or no questions like "It is less stressful if all waste is disposed in one waste bin receptacle". The survey was sent out to students of whom 59 filled it out. From these results we learned that only 10% of students feel strongly confident that they dispose of waste in the correct receptacles. As well, 62% of students say they use the other bin if the recycling or landfill one is full. And 60% of the students are happy if they are not expected to make any decisions while disposing of waste. This led us to conclude that most students are not aware of the right steps to dispose of their waste properly and is a large issue.

We also talked with some of the janitorial staff in charge of the LSA building on campus along with the Union. These informal conversations were meant to get the main ideas of how one goes about changing the bags on the trash cans as well as any downfalls to the current designs. On the janitorial staff carts that are used to carry their supplies, they currently have three different sized bags for the most common trash cans on campus. We wanted to make sure that we don't add a different bag from anything they already carry. As well, they service a lot of different locations and trash cans in one day. Therefore the interaction that they have with the trash.py bin should not add substantial time to their route. Having the bins accessible and the system easy to use and understand is crucial in order to not make their jobs any harder.

Design Brainstorming

In order to explore the design space for our solution we started off with large brainstorming sessions as teams to develop different concepts. Some example concepts from these sessions can be seen in figure 2 and 3. We wanted to make sure that we were able to come up with the best designs possible but get creative. As we went through different iterations of brainstorming such as small individual, group, and design heuristics, we ended up with around 45 different design ideas for the trash.py bin.

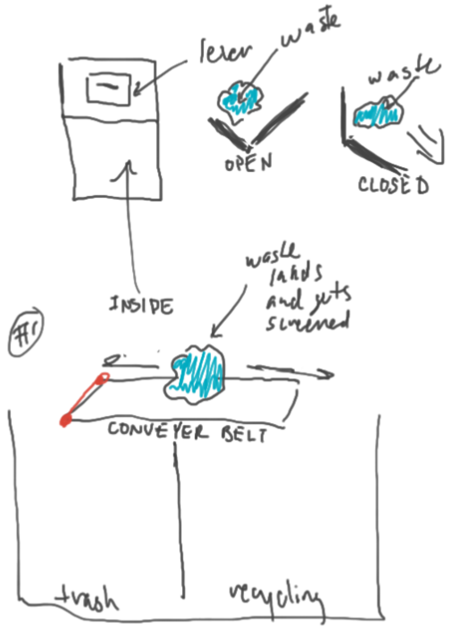


Figure 2. Example concept #1 showing an open latch deposit system sling with a converter belt that the waste would fall onto. The conveyor belt would then move left and right depending on the decision made by the system whether it was trash or recycling.

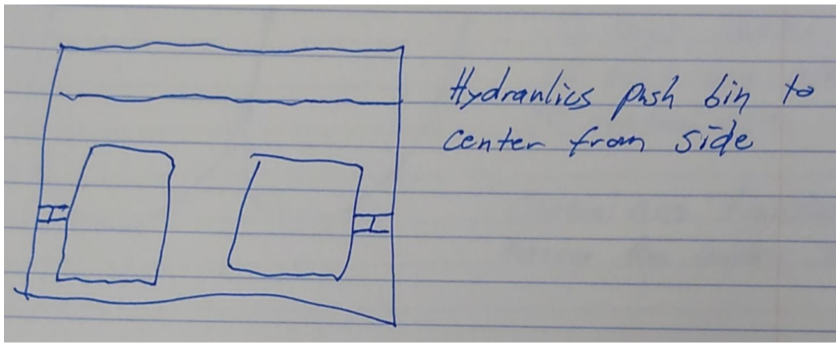


Figure 3. Example concept #2, showing two bins that are then pushed into the middle of the bin in order to receive the item thrown away.

Initial Filter. The first filtering step was a gut check for ideas that were not possible in the time frame of the semester-long project as well as things that couldn't be done within our current knowledge and expertise. This then led to the morphing of ideas and left us with 10 final design concepts that encompassed all the unique ideas we had come up with. For the concepts, the main goal was to make sure that the design was not limited but that we kept some basic ideas in check. This is when the filtering criteria became important.

Final Filter Criteria. The criteria used was robustness, ease of use, complexity of parts, and affinity for taking pictures. Each of the criteria was then given a weight based on the importance of meeting this criteria which ranged from 1-5. Robustness, which we took to mean the robustness of the sorting mechanism ensuring less failure, was given a weight of 5 due to its inherent relationship with the bin completing the task. Ease of use, which we took to mean how intuitive was it for someone like the janitorial staff as well as the students to use on both the front and back end, got a weight of 3. Complexity of parts got a weight of 2 due to being important in the context of the time frame we have as well as wanting to keep the design as simple as possible. And affinity for taking pictures, which we took to mean how likely will the design result in a good quality image for the system the majority of the time, got a weight of 5 since without a good clear picture, the system won't be able to make the correct distinctions. From these criteria then each design was given a ranking of 0-3 in regards to how closely they meet the criteria. With 0 meaning the criteria is not met/supported and 3 meaning that it is strongly supported. As a team we ranked the final 10 designs in each of the criterias leading to a unanimous decision for the final design.

Software

After designing the trash.py bin, we needed to design a system of software packages that the bin would use to perform its tasks. We knew that the bin needed to be able to operate mechanical systems, which necessitated a bin operating system. We also knew that the bin had to be able to classify waste with a camera, so a computer vision system was needed. We decided after some brainstorming that all of the computation should be done separately from the bins via the cloud, minimizing the computing systems in the bins and lowering e-waste. This decision, however, meant that there was an additional need for a cloud-based server system to communicate with the bins and perform computer vision tasks. Over the course of the semester we were able to implement both the web server and the computer vision model, but did not create a bin operating system because we did not build a fully-functioning bin and did not know what features the operating system would need. In the future, an operating system for the bin would need to be developed.

Computer Vision. The problem of extracting objects from images is called Computer Vision (CV). Our goal for the trash.py CV system was to be able to have a computer classify an image as either trash or recycling. This is typically accomplished using convolutional neural networks (CNNs). A CNN is a function that takes an image's data as input and gives data as output. The data and its protocol can be arbitrarily designed by the programmer, but it typically consists of a vector of numbers, each from zero to one, representing the CNN's confidence that the input image contains a certain class of object. For example, if the CNN is confident that the input image is of an object of class 0, the output vector might look like [0.99, 0.01, 0.00].

CNN Architecture. Both CNNs and regular neural networks perform their tasks of turning inputs into outputs through a series of layers. A layer is an object that takes in an array of inputs, performs some transformation on those inputs, and outputs said transformation. When training data is passed through the network, the outputs are compared to the correct outputs and a gradient descent is performed on the weights and biases of the layers to minimize an objective function. The objective function is usually the sum of some loss function with respect to the outputs and the magnitude of the weights [2]. Figure 4 gives an example of a loss function commonly used known as binary cross entropy

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

Figure 4. Formula for binary cross-entropy. y_i is the truth value for class i, and $p(y_i)$ is the model's predicted value

A CNN differs from a traditional neural network because it contains Convolutional Layers, or convolutions. Instead of just performing a matrix multiplication on the input data, a convolutional layer contains a series of filters. Filters are matrices that operate on small portions of the input, moving across the input a specified step size/stride, and performing a matrix operation at each step. The resulting output diagrammed in figure 5 is the concatenation of the outputs from all of the filter operations [3]. During training filters generally become grayscale images which represent certain features of the objects we wish to classify. For example, a filter in a network meant to recognize dogs may begin to look like a nose or ear after training.

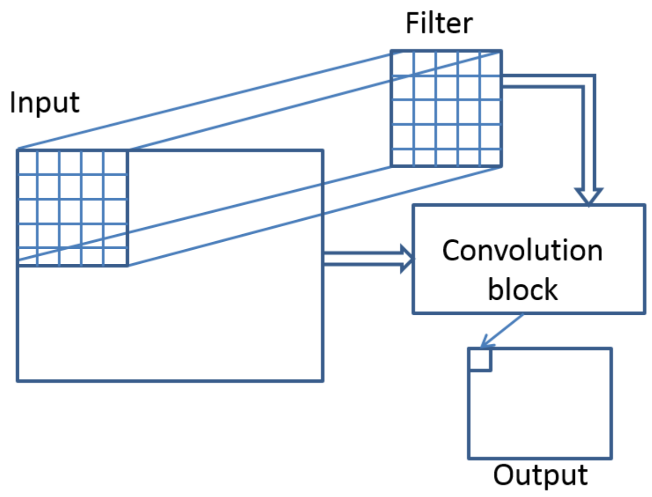


Figure 5. Visual explanation of a convolutional filter operating on an input and producing output [3]

The advantage of the convolutional layer is that the target object can be anywhere in the image and the filters will operate on all parts of the image. During a convolution the filter will be matched up with the object in the image at some point. Conversely, in a traditional neural network an image that was correctly classified by the model would likely be classified incorrectly if the image were to shifted, scaled, or rotated.

Data. A CNN requires a large amount of data for training. This data comes in the form of input images and output labels. For a prototype of our model, we opted to not generate our own training dataset. Also, the image-taking portion of gathering data would be done automatically for us if we installed a working prototype on campus. In order to find a good pre existing dataset, we first defined our CV problem to be as follows:

Input images are taken from the camera in the trash.py bin. The camera is stationary so the distance from the holding platform and the camera is constant. The background is a v-shape to hold the waste items, and is all white. The camera has a flash installed to consistently illuminate the scene. There is only one object in each image and the object is either trash or recycling.

From these invariants of our problem we were able to define a set of criteria for our training dataset. We wanted our dataset to be a set of labeled images of public bin waste taken relatively close up at a top-down angle against a white background with flash. Luckily for us, we found the TrashNet dataset.

The TrashNet dataset contains 3000+ images of trash and recycling. The images are relatively close with flash against a white background. The dataset differed from our specification in that the images were not taken from directly overhead and were instead at an angle, but we decided to use it for our model prototype anyway as the images were good enough to establish a baseline. Figure 6 gives an example of an image of a food wrapper from the TrashNet dataset.

Figure 6. Example of a food wrapper labeled as trash from the TrashNet Dataset. Consistent with the rest of the images, the photo angle is not top-down.

The problem that the TrashNet dataset was created to solve was slightly different than trash.py's. The classes of the dataset were allocated such that the number of images for trash was equal to the number of images for each category of recycling. This resulted in there being very few images of trash compared to recycling. In ML, this is known as a class-imbalance.

Class Imbalance. As shown in figure 7, there were far too many images of recycling compared to trash. This created a problem for training a neural network, as a model trained on this imbalanced data could learn to just guess that every image is recycling and get a greater than 90% training and validation accuracy. Such a model wouldn't actually learn the problem that trash.py is trying to solve, and in practice would default to throwing all items in the recycling bin.

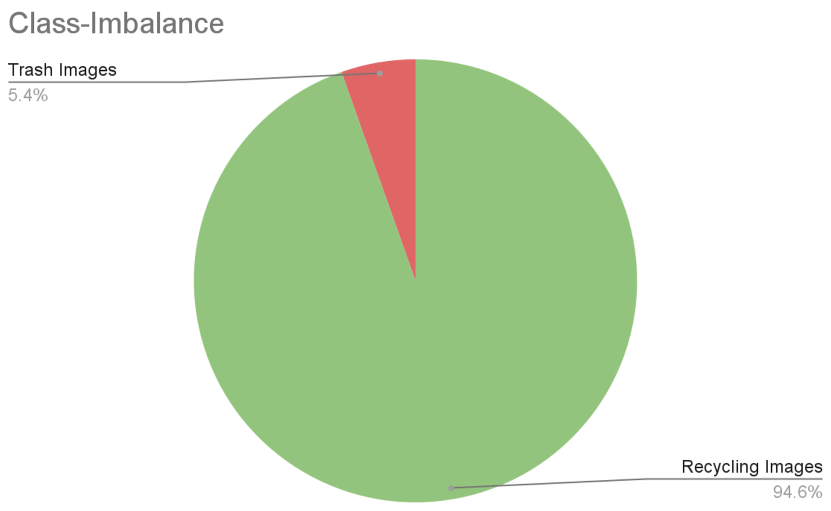


Figure 7. Pie chart showing class-imbalance of the TrashNet dataset for the trash.py CV problem.

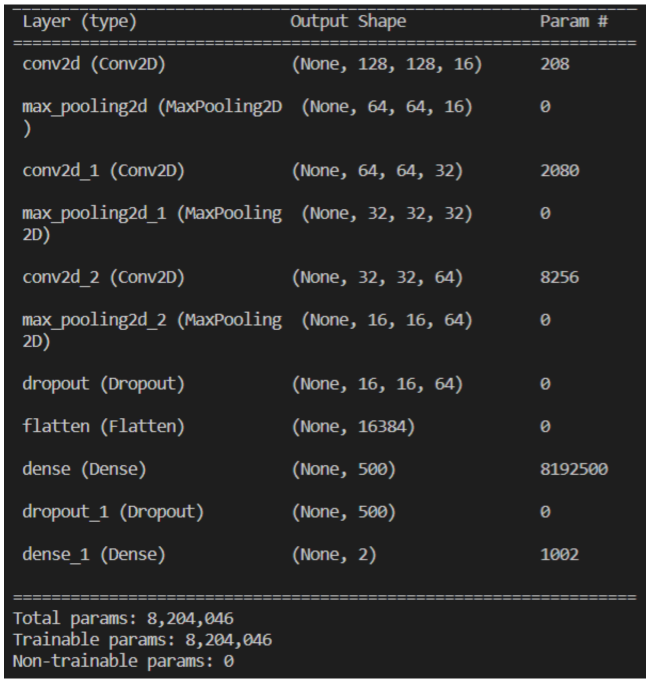
There are a few methods that can be utilized to combat the class-imbalance problem. The following are from [4]

Random Over-Sampling (ROS) is the process of adding bias toward the under-represented class (UC) and away from the over-represented class (OC) during training. It is accomplished in the training function. When creating a batch of training data, instead of choosing randomly from the entire training set the function will choose an equal number of each of the classes. This results in the classes being balanced while training despite the dataset being imbalanced ROS does require more training time, however, as there is effectively more data to train on.

Random Under-Sampling (RUS) is the counterpart to ROS. Instead of adding more of the UC to each training iteration, it adds less of the OC, undersampling from the OC. The class balance during training is the same as ROS, but training takes less time. This method is also less liable to overfit the model, as it isn't being trained on several identical copies of the UC.

Data Copying and Data Deleting are methods similar to ROS and RUS, but instead of being implemented in the training algorithm they involve making physical changes to the dataset. These methods are faster to implement, but take more storage as a copy of the original dataset should always be saved separately. The new datasets either contain additional copies of the UC or fewer images from the OC, respectively.

trash.py opted to use the Data Copy method to address the problem of Class-Imbalance, as there are additional methods available to this solution that can make it more powerful. The copies of the UC were not identical copies to the original trash data. Instead, each of the fifteen copies of the trash images were rotated by a multiple of 24 degrees using OpenCV . This made each image of a specific trash item unique to prevent overfitting. Figure 8 shows the architecture for the trash.py model prototype.



```
______________________________________________________________
 Layer (type)                Output Shape              Param #
==============================================================
 conv2d (Conv2D)             (None, 128, 128, 16)      208

 max_pooling2d (MaxPooling2D  (None, 64, 64, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 64, 64, 32)        2080

 max_pooling2d_1 (MaxPooling  (None, 32, 32, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 32, 32, 64)        8256

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 dropout (Dropout)           (None, 16, 16, 64)        0

 flatten (Flatten)           (None, 16384)             0

 dense (Dense)               (None, 500)               8192500

 dropout_1 (Dropout)         (None, 500)               0

 dense_1 (Dense)             (None, 2)                 1002

==============================================================
Total params: 8,204,046
Trainable params: 8,204,046
Non-trainable params: 0
```

Figure 8. Output of calling model.summary(). There are three convolutional layers that are pooled afterward, followed by dropout and dense layers for just over eight million trainable parameters.

Cloud server. trash.py opted to use an external server to host the model, rather than host the CV system on the bin. There are several advantages to this method, and some disadvantages. The main advantage is less energy and waste. If the bin doesn't need to do mass computation on site, then it can be outfitted with less powerful onboard electronics and draw less power. This is more sustainable as less e-waste is generated per bin. There is also more modularity with this approach. If the computation is abstracted to the cloud, then the bin's interface with the cloud only includes sending image data (along with some metadata) through a web api and receiving a classification decision back. The actual model can then be altered at will separately from the bin and no change is necessary to the bin itself. A new model can also be trained faster on the cloud than it could on an onboard computer. trash.py could also theoretically host multiple models on the cloud that correspond to multiple recycling schemas for bins in different locations.

There are some downsides to the cloud approach, though. There is additional latency in the system as the bin has to make an API call and wait for a response. We believe that this latency is mitigated by the cloud's faster computation time. The bins also need to have wifi connectivity, and cannot function properly if they lose connection. We believe this can be fixed if the bin also keeps a copy of its current model onboard as backup. In the case of a wifi crash, the bin could still perform its task, just slower.

We built a flask server in python to act as the brain for the trash.py bins. The server is relatively simple, and only contains necessary functionality, such as model training, data collection, interactive web applet, and the trash.py API. When the server receives a call from a bin, it takes image data as input, and returns a classification. Flask is a python framework that runs a server that along with sending and receiving API calls, can also serve templated html pages to a web browser. We utilized this functionality to show how the model works by developing a web applet that interacts with the model and camera.

Web Applet. Because trash.py did not yet build a fully-functional prototype bin, we needed to have a way to test the web systems and interact with the model. The web applet we developed does just this and simulates what goes on in the bin. It combines the functionality of the web server, model, camera, and staging area prototype (figure 9). On the landing page (figure 10), the user is greeted with the trash.py logo and has the option to click a button to classify an image. There is also an optional filename input box for trash.py internal testing.

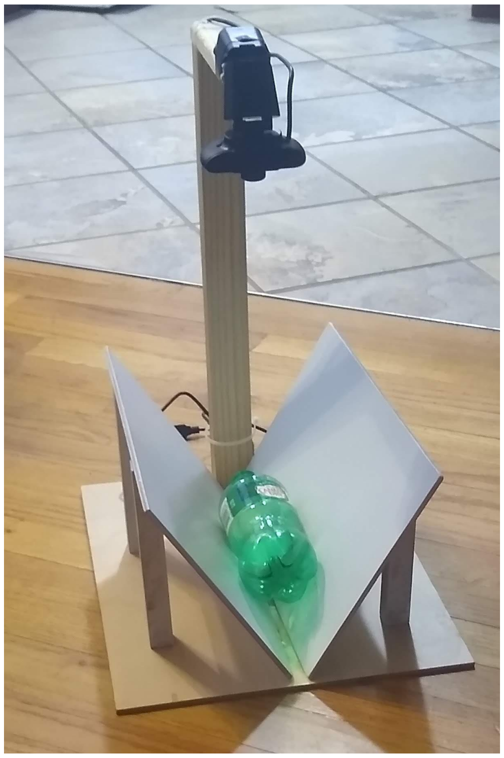


Figure 9. Prototype of the trash.py staging area. Items are held in the v-shaped area to be photographed by the camera, which points down from the top. The camera connects to a computer running the flask web server to simulate the bin's functions

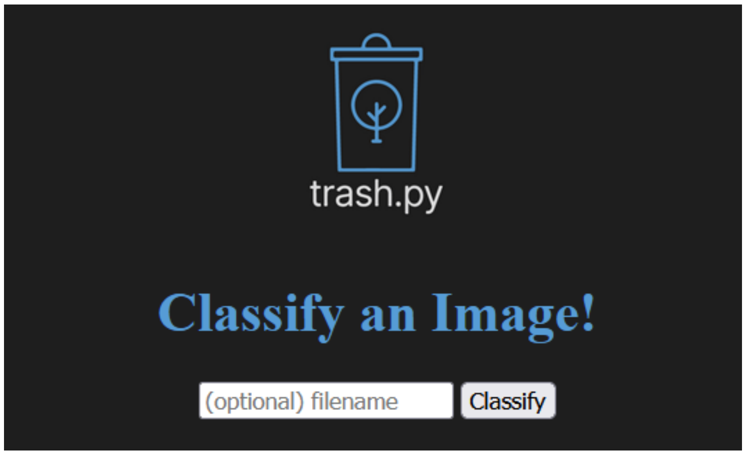
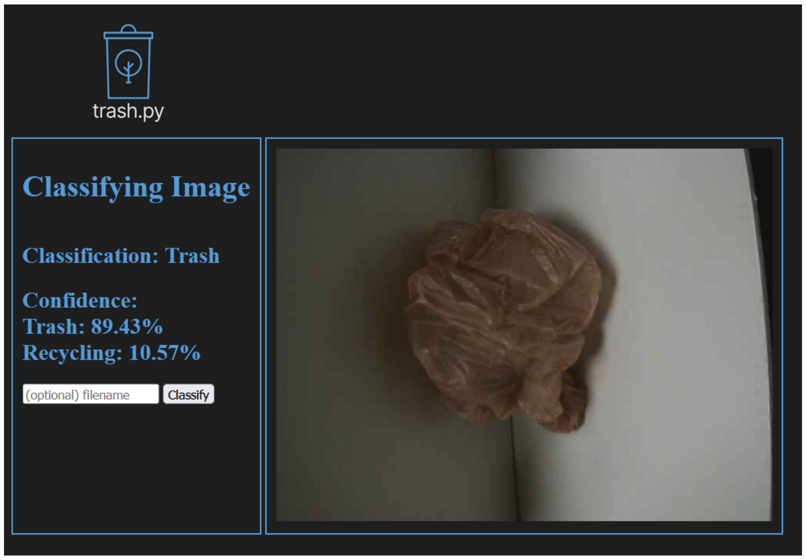


Figure 10. Landing page for trash.py web applet. If no filename is given, the image is saved in the database as item_default.jpg which is overwritten. Giving a unique filename allows the image to persist in the database.

When the user clicks the 'Classify' button, the camera takes an image of the staging area. Next that image data is sent through a post request to the web server, which then feeds it through the model. The server then interprets the model's output and returns the classification of the image back to the browser for display. The browser then redirects to the display page (Figure 11), and shows the user the image taken, the classification of the image, and the confidence values for each of the classes.



Classification: Trash

Confidence:
Trash: 89.43%
Recycling: 10.57%

Figure 11. Classify page. The image displayed is the real time state of the staging area prototype when the classify button was clicked

The web applet acts as a simulation of the actual bin, with the web browser and camera/staging area acting as the bin to interface with the server. It does well to demonstrate the system working. If the user wishes to classify another item, they can exchange the items in the staging area and click on the classify button again.

RESULTS

Final Design

Our final design, as seen in figure 12, for the trash.py bin has an open latch deposit system that intakes the item and deposits it into the staging area. The staging area is where the system takes the image of the item thrown into the bin to then run through our model and conclude whether it is trash or recycling. Therofere, the system then goes through two scenarios. In scenario one as depicted in figure 13, the system concludes that the item is recyclable and the right latch swings open to deposit the item in the recycling bin. In scenario two as depicted in figure 14, the system concludes that the item is trash and the left latch swings open, depositing it in the trash bin. The open latch has a slight angle to the opening meaning that whatever is thrown into the receptacle won't fall into the staging area unless the latch is closed all the way. The time that the system takes to classify the image as well as move the flaps open and closed on the desired side is the time eaten up by having to fully open the latch and close it all the way. This inherently reduces the amount of time that the user feels like they are standing around for the bin to work. The only requirement here is that the user throws one item in at a time so that the image depicts one item and it is sorted out correctly.

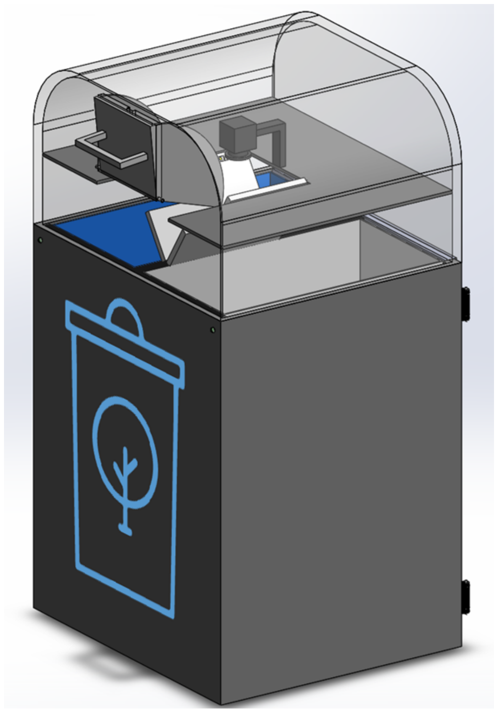


Figure 12. Final design concept for the trash.py bin.

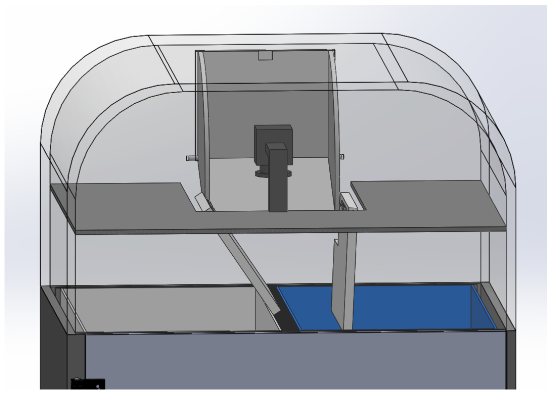


Figure 13. Scenario 1 in which the item detected by the sorting system is recyclable

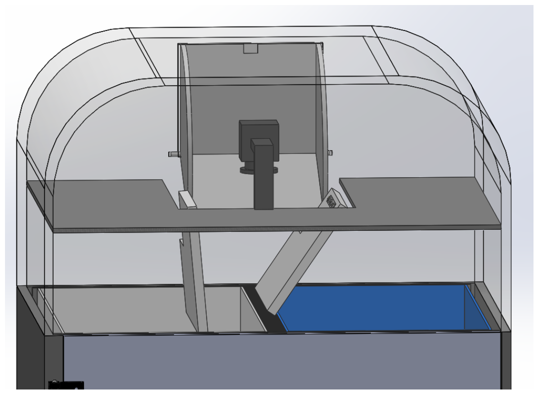


Figure 14. Scenario 2 in which the item detected by the sorting system is trash

Design Features. The solution is also fitted to the existing michigan bins in order to allow for easier integration and use of items that the university already has available. In the conversations we had with the janitorial staff, they already carry the right size bag for the michigan bins found in figure 15. By using these bins in the design we use resources already allocated at the university and don't increase plastic consumption. As well as making the integration for the janitorial staff equipment as smooth as possible.

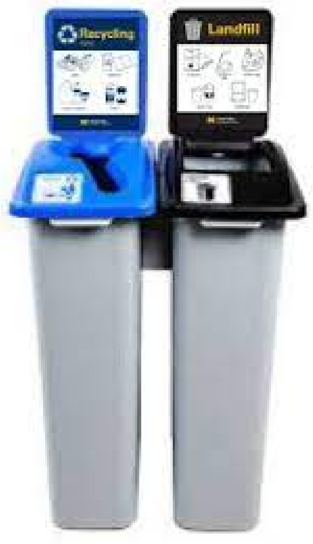


Figure 15. Current bins found on the campus of the University of Michigan that fit into the trash.py bin.

Since the design has a top, the best way to change the bags on the bins is to pull the bins out and change them where there is room to remove the bags up and out. To make this simpler, the design has a slide out rack on wheels that holds both of the bins and can be seen in figure 16. This allows the staff member to easily pull the bins out no matter the weight of the trash inside. Then changing the bags takes on the exact same process as the bins used everywhere else on campus.

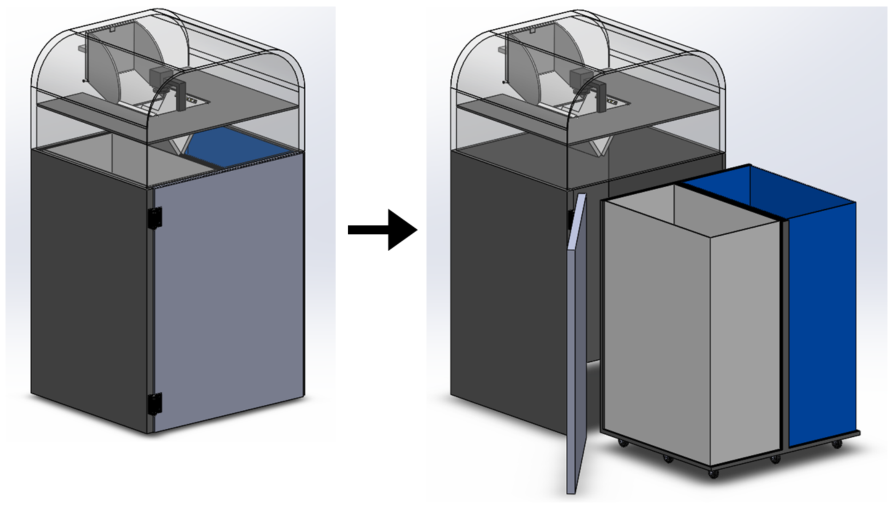


Figure 16. Door on the back of the bins which after opened reveals the slide out rack for easy removal of filled bags.

LED lights were also added to the front of the bin in order to help signify the status of the bin. The lights can be seen depicted in figure 17 below. If the lights are green, that would signify that the bin is working properly and isn't in need of any service. Orange would signify that the respective side is full. Meaning that if the left side shows orange then the recycling bin is full and would need replacing. And lastly, red would signify that the trash.py bin is not working properly and is in need of maintenance. By having the LED light, it allows for an easy check of the bins status at just a quick glance.

In order to know if the bin is full and needs to be changed, we added a weight sensor to the slide out rack and can be seen in figure 18. Weight can vary when it comes to trash and recycling and therefore the sensor is set to different weights for fullness detection. Other systems like the trash.py bin have used weight sensors before in order to know when the bins are full. This is something that would have to be tested to see if it is reliable information. If not, a sensor can be added to the bins to see how high the trash/recycling is piling up. We wanted to first stick to weight sensing to make sure that bins inside would not have to be modified so that we could keep the use of the Michigan bins. The detection of when the bin is full would be paired with an app that would let the janitorial staff know that the bin is at capacity. For each building's staff handling the waste, one would be able to choose the required campus and the building to reach their own setup. Each building could have views showing the real-time status of all bins a staff member is responsible for, along with statistics comparing recyclables vs general waste in the bins.These are some basic features of the app that will keep evolving in the future.

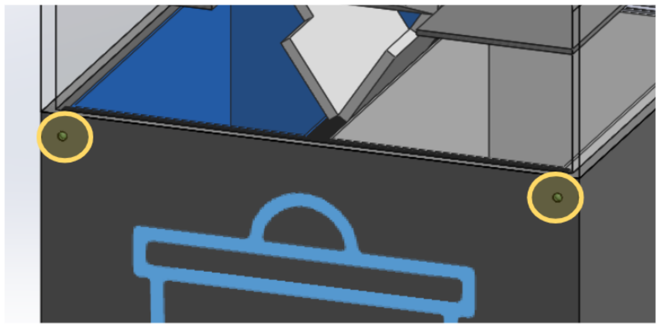


Figure 17. LED lights found on the front top corners of the bin to help signify the bins status.

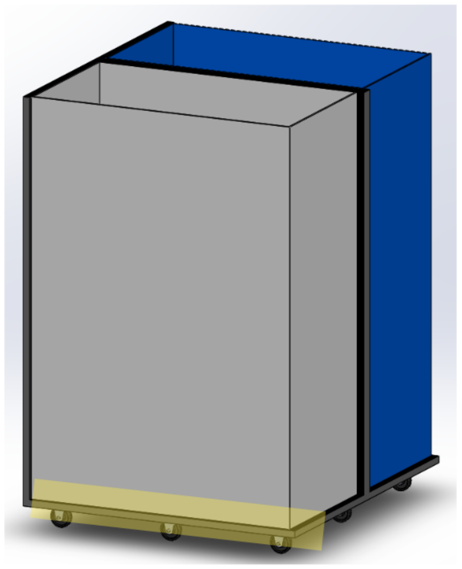


Figure 18. Slideout rack on the trash.py bin which features a weight sensor on the bottom between the bin and the holding rack.

Model Performance

Figures 19 and 20 show the accuracy and log loss of our final trained model, with a final test accuracy on the TrashNet data of 98%.

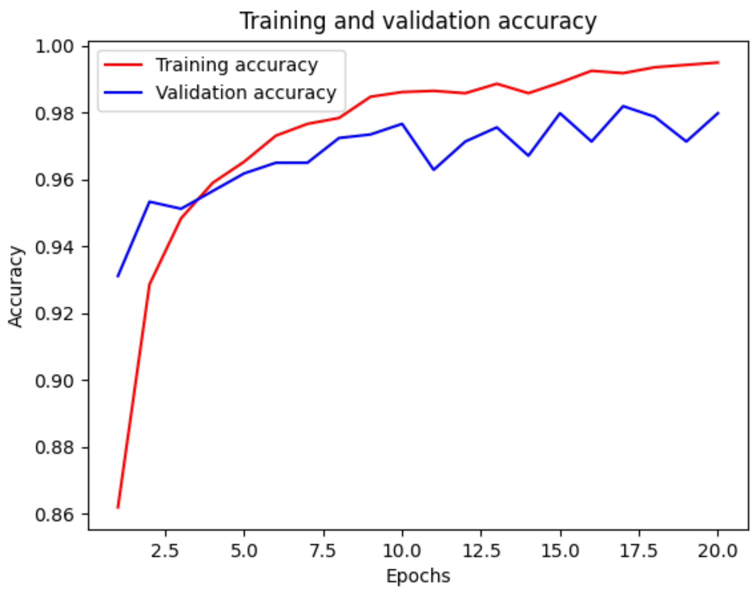


Figure 19. Graph of the training and validation accuracy at each epoch.

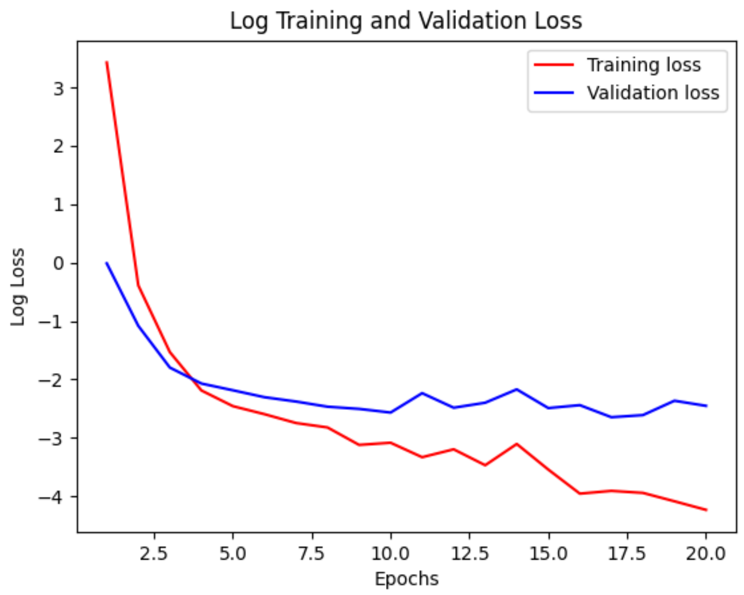


Figure 20. Graph of the log of training and validation loss at each epoch.

Impact

As mentioned earlier, around 40% of waste in UMich bins is classified incorrectly by the users. Our model has a very high accuracy, and we have reason to believe that a working prototype would have a high accuracy when sorting actual waste. Thus, given that the university campus produced about 9,000 tons of waste in 2021, we estimate that with trash.py systems we could intercept over 3,000 tons of waste annually that previously would have ended up in the wrong bins. This has a widespread impact, as more recycling can be sent off to MRFs, less time and energy can be spent sorting through recycling at MRFs, and more CO_2 can be saved as a result of this increased recycling.

DISCUSSION/CONCLUSION

What we learned

Throughout the project the need for change of the recycling and trash system at the university was largely emphasized. There is a large margin for improvement when it comes to how we deposit waste at the university and how we can reduce this number. We chose one very specific way to tackle this issue but there are many other ways to make this change. As the world progresses we need to make sure that we are using the new technology and knowledge to better the environment and help reduce the amount of waste going into landfills. Focusing on issues of reducing all sort of single use plastics and other waste would help reduce the amount of waste needed to be sorted in the first place.

Proof of concept

The trash.py project has shown signs of viability, but more problems still need to be solved in order to prove that trash.py is a viable concept. The web systems and camera all work correctly, but the embedded system of microcontrollers and microprocessors still needs to be built and tested. We anticipate it would take a lot of work and some design changes to build a fully-functioning prototype, but we believe that the trash.py concept is sound and can be viable

What we would change

The project was successful in terms of what we hoped to learn and accomplish in the short time span that we had. If anything were to change it probably would have been getting in contact with OCS a lot sooner. Going forward in the project we learned how valuable their input and data was in understanding not only the problem but the behavior of people's relationships with trash cans. Some of the design ideas could have been influenced by more pointed data on exactly what goes through a normal bin in one day to make sure that edge cases have been taken care of. In terms of changing things in our actual design the main area of focus would be the staging area. As far as we have concluded it should work within its intended purpose but there are things we didn't consider until later on. For one, any liquids that were to be tossed into the bin would probably contaminate the recycling by leaking through the flaps forming a v shape. This is something that would have to be redesigned for that purpose. As well, thinking about very large waste items and what to do with them. If it is ok to limit the size of the items being thrown in like we have, or if this will actually cause a bigger issue.

Moving Forward

We would like trash.py to be more than just a bin that separates trash from recycling. According to the OCS [5] the chief opportunity for a more sustainable campus is in the form of composting. We would like to add the ability for the trash.py system to handle compostable material and divert it from the trash/recycling stream. Additionally, were trash.py adopted by the university, students would no longer need to know the difference between trash and recyclable materials on campus, but would still produce and dispose of waste off campus, possibly incorrectly. This is why we would also like to partner with the university in the future to design and employ a waste classification education campaign in order to make sure all waste is disposed of properly by everyone in the university no matter where they are.

REFERENCES

[1] B. Rose, “U-M Waste Bin Checks Tracking February-April 2022.” Michigan, Ann Arbor, 2022.

[2] C. Green, “Cross Entropy,” *Cross entropy*, 2016. [Online]. Available: https://heliosphan.org/cross-entropy.html. [Accessed: 24-Apr-2022].

[3] M. Panwar, “(PDF) modified distributed arithmetic based low complexity CNN Architecture Design methodology,” *ResearchGate*, 2017. [Online]. Available: https://www.researchgate.net/publication/321237881_Modified_distributed_arithmetic_based_low_complexity_CNN_architecture_design_methodology#pf2. [Accessed: 24-Apr-2022].

[4] Y. Wu and R. Radewagen, “7 techniques to handle imbalanced data,” *KDnuggets*, 2022. [Online]. Available: https://www.kdnuggets.com/2017/06/7-techniques-handle-imbalanced-data.html. [Accessed: 24-Apr-2022].

[5] “Environmental metrics,” *University of Michigan - Office of Campus Sustainability*, 02-Nov-2021. [Online]. Available: https://ocs.umich.edu/resources/sustainability-data/environmental-metrics/. [Accessed: 24-Apr-2022].