Clean Planet Project
Field Session Summer 2021

Client:
Josh Rands

Team Members:
Dakota Brown, Jonathon Robel, Logan Daigler, Michael Alvarez, Steven Bullen

Date:
June 16, 2021

Introduction

Our client for this project was Clean Planet Project Co. which is a Colorado based 501(c)(3) nonprofit startup utilizing innovative solutions to make the planet a cleaner place. They are a fully remote and part-time team made up of passionate and fun individuals who are working together to implement creative solutions that make being eco-friendly more fun and accessible for all. To accomplish this they have developed an app called Litter CleanUp, which is a crowd-sourced trash cleanup platform that makes picking up litter fun. The app currently has over 350 users in 20+ different countries who have picked up 15,000+ pieces of litter. The app was built using Google's Flutter mobile app framework and Firebase platform. A major part of the Litter CleanUp app is the approval of litter submissions, a group of images composed of a number of litter photos and one trash can photo. To ensure that the user's submissions are truthful and contain litter, submissions must be verified which is currently a 100% manual process.

Problem:

The current system that the client is using requires the manual verification of submissions. This takes large amounts of time and manpower as someone must check the database for new submissions and then download them and verify. As it currently is the client is unable to expand and advertise his business as he would be unable to accommodate the increase in submissions.

Our Goal:

Streamline submission verification within the Litter CleanUp application, reducing the requirement for human verification. This will speed up the process so that the application can scale more easily and become more widely used. Accomplishing the larger goal of cleaning the world by removing litter.

Definition of Done:

The client needed a model that could accurately identify a valid and invalid submission using Tensorflow. The model would need to be able to identify certain aspects of an image that would make it invalid such as being inside or people in the image. A related goal to that was to create some backend logic that would use the model output and confidence to send unconfident results to manual review. Our other stretch goal was to integrate our machine learning algorithms into his already existing application as an API into the Firebase platform. To be fully integrated, our algorithms would update user scores and tell the user in the app whether their submission was Valid, Invalid, or sent in for manual review. However, this was subject to change as the client is looking into alternatives to his existing system.

Requirements

Functional Requirements

- Machine learning model to categorize submissions as verified, denied, or unknown.
    - Able to recognize if the image is from inside or outside.
    - Able to recognize trash cans or trash bags.
    - Able to recognize humans.
    - Able to recognize litter on the ground.
- Accurate to a sufficient level that it cannot be easily exploited.
- For submissions with many photos, take a random sampling from the provided images.

Non-Functional Requirements

- Image data must not take too long to compute (large volumes of images pushed through a machine learning model takes time and resources).
- Machine learning model(s) must run locally (offline).
- Must be able to integrate with the current application as an addition.

Features suggested by the team:

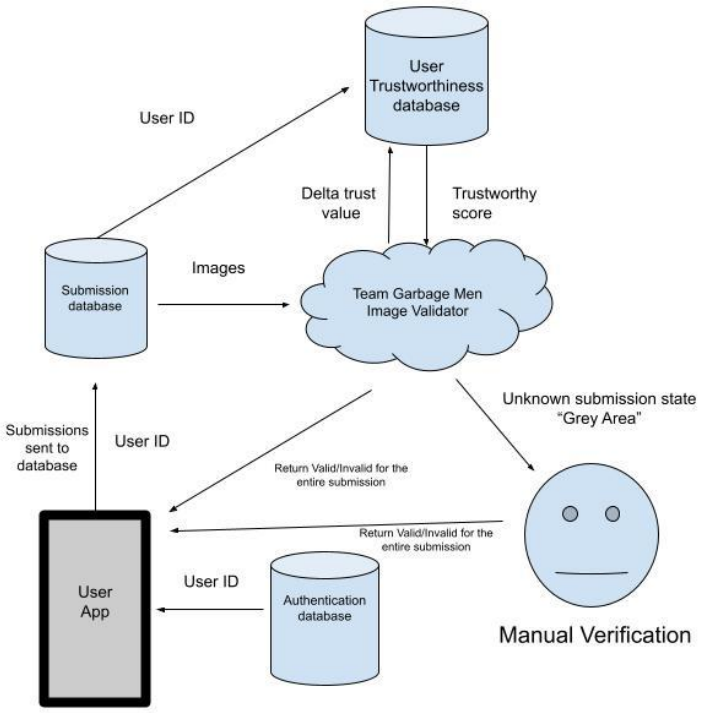
- Grey area for submission that the model can not exactly identify.
- Trust level for users to sample less images and minimize required processing power.

System Architecture

Application Workflow:

We have two primary architectures related to our work. First is the application architecture and how it interacts with our product as below in figure 1. The workflow begins when a batch of images, referred to as a submission, is created by the user and sent to the submission database. Once that submission has been fully uploaded the respective user trustworthiness score is retrieved from another database. Both the images and the trustworthiness score are used to feed selected images into our automated validator composed of two distinct ML models. To understand the process further, a trustworthiness score is an indicator of how trustworthy a user is. With all users starting at a score of zero, it increases with valid submissions and decreases with invalid submissions. We use it to decide how many images in the submission we will sample and feed to the models. So for example a user with a low trustworthiness score may have all their images analyzed by the model while a highly trusted user will only have 10% analyzed.

Next, if the validator is confident in its choice, then we send the results immediately back to the user and also update the trustworthiness score of the user; otherwise it will go to manual verification as an unknown state. Manual verification will then send out the results in the exact same way as the automatic validator. The level of confidence necessary to bypass manual verification is a tunable parameter inside the image validator component and will be mentioned in more detail shortly.

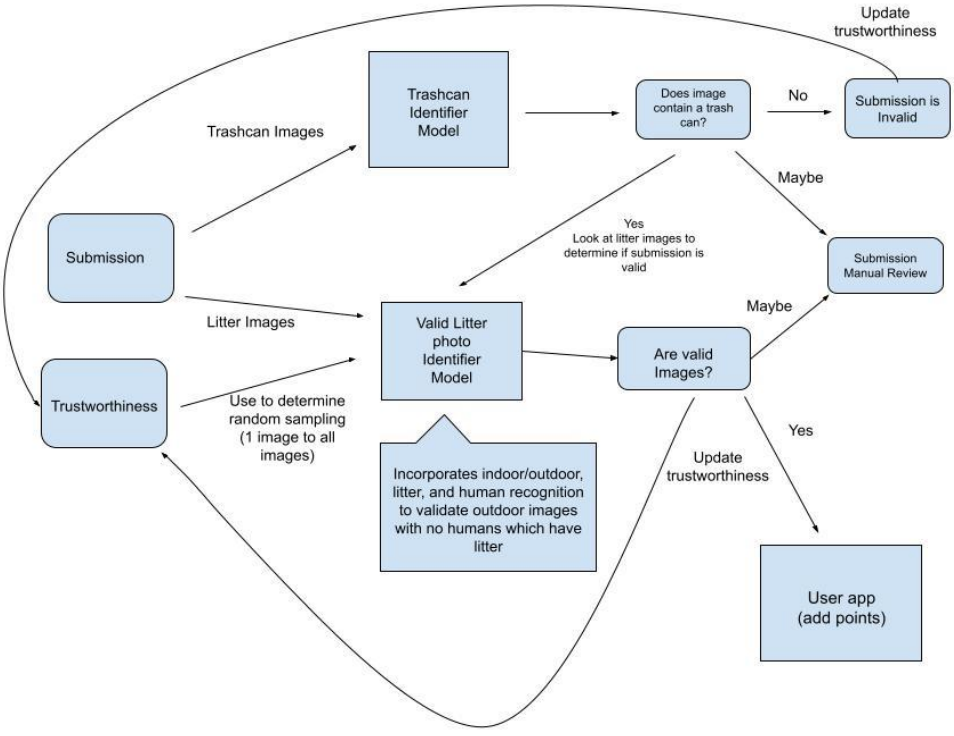


Application Workflow - Fig 1

Validation Model Workflow:

The other architecture related to our work is that of our automated validator as seen below in figure 2. This architecture begins with two inputs, the group of images known as the submission and a user trustworthiness score. From there the trash can image is taken from the submission and sent into the trash can identifier model. If a valid result with sufficient confidence is achieved, then random samples are pulled from the rest of the submission based on user trustworthiness. These selected images are sent to the Litter Identification Model for further verification. If they all return valid with sufficient confidence the submission is then deemed valid and the process moves on. If at any point in this process an image returns invalid with sufficient confidence the workflow is interrupted and the entire submission is deemed invalid. Likewise, if a result with low confidence is returned then the entire submission is sent to manual review. The validator returns the state of the submission but based on that state it also returns an updated user trustworthiness score.

As mentioned before sufficient confidence is a tunable parameter hidden inside this workflow. It's composed of two values, one for valid confidence and the other for invalid confidence. If either model returns a valid label then the valid confidence value is used as a threshold with invalid labels using the other value. For example if a trash can image is determined to be valid with 91% confidence and the confidence threshold is at 60% then the image can be labeled as valid. Now if the confidence were lower, something like valid with 56% confidence then the image would be labeled as unknown and the submission would have to be manually reviewed.



Validation Model Workflow - Fig 2

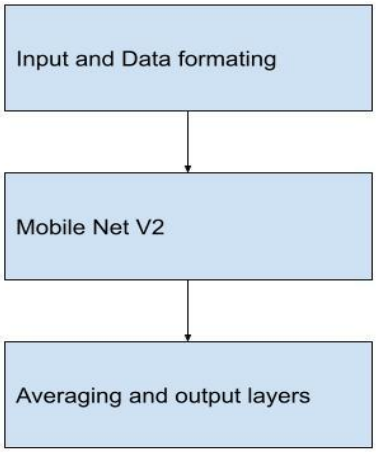
Technical Design

Transfer Learning with Mobile Net V2:

The best performing model that was created used transfer learning on the Mobile Net V2 (MNV2) model created by Google. MNV2 is a large image classification model created as a research project at Google. It is available through TensorFlow so it integrated well with the systems we had already been testing and creating. MNV2 has approximately 2.2 million pretrained parameters. Google trained MNV2 on 1.4 million images so it is able to identify a wide range of different features, this makes it perfect for a transfer learning model.

How transfer learning works:

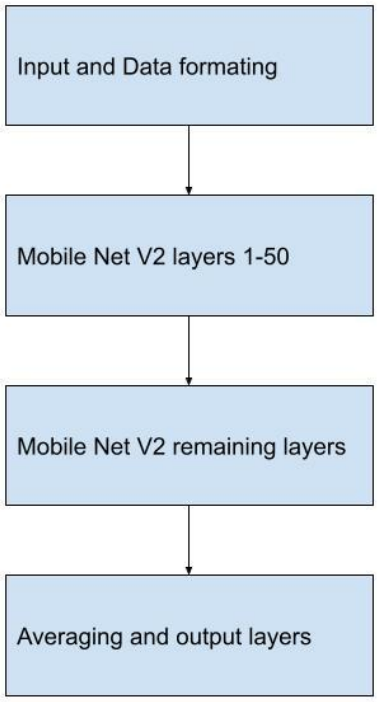
Transfer learning is a way to use a very generalized model and make it more refined to a specific task or in our case a specific image classification. It does this by putting layers around the generalized model that can then be trained for our specific needs. By freezing the layers within the base MNV2 model, we can train parameters around it to make them work for the labels we need out of our model. Mainly in the averaging and output layers we train the parameters to match our labels.



Transfer learning - Fig 3

It is also possible to unfreeze a portion of the layers within the MNV2 to allow them to be trained on our data as well. This allows for even more tuning and precise use towards our

specific application. It does this by allowing more parameters to be trained on our data rather than being frozen to their original values and still keep a majority of the base model parameters generalized.



Transfer learning with Training base model layers - Fig 4

Since the MNV2 is over 150 layers deep it still retains a large majority of its original training that the parameters had.

Using this led to a much better jump in accuracy of predictions by our model without having to create a model as detailed as the MNV2 from scratch. Creating a model of that amount of detail would have taken much more time to complete and would not necessarily have been any more accurate than the transfer learning approach. On top of that the amount of images fed to MNV2 was just improbable for this project as finding and handling that amount of data onto our platforms would have caused more issues and taken more time than it was worth.

Quality Assurance

Multiple models to test against and find the best one:

In order to ensure that our client receives a model that is not only accurate, but also dynamic, we constructed and tested several models against identical datasets in order to breed the best one. We used completely different architectures of models to help see how they could compare. One used a transfer learning approach pair with the Mobile Net V2 and the other was a more simplistic Sequential model that works for much more basic image classification.

Testing tunable features such as grey area and confidence:

Since models will never be 100% accurate so the option for adjustable parameters for fine tuning confidence and grey area thresholds. This helps to reduce the misclassification of submissions. The grey area is when submissions are sent to manual review, as the model has a low confidence of which label it belongs to.

Validation data sets for the model:

Validation data is used to show how the models will perform on data they have not seen and give an idea of the true performance. Continual testing can be done on new or updated validation datasets by running all models against the same validation data to choose the best performing model. As some data was created by the team a separate set of real world data was kept as a validation set for final testing of models for comparison.

Metrics for Models:

For a majority a simple accuracy metric was used to test how the models performed on the validation and training data sets. After seeing a very high accuracy score to start even with simple models we used other metrics to ensure that we weren't getting a false idea of a well performing model. The most important other metric used was a confusion matrix showing how many images were misclassified and how they were misclassified. This showed if it favored giving an image a valid or invalid classification more.

- Other Metrics
    - Precision
    - Recall
    - Mean Squared Error

Hello world smoke test (Firebase Integration):

To ensure that we have integrated with Firebase, we have built a simple program to show that entries within the Realtime Database can be accessed and changed as intended.

Database test to ensure correct integration with other systems:

Test that to ensure trustworthiness and point values are correctly updated on a Valid or Invalid submission

Github VCS for cloud functions:

Used to manage code branches and code versions to ensure that all members can work on code that is up to date and to manage merge conflicts

Firebase testing function:

Firebase has an automated testing service that runs through all of the parts of the app to ensure they are working

Code reviews on firebase cloud functions:

Once deployed onto firebase they are unviewable but through Github we can read the code that was deployed.

Full integration testing:

The client will help with integrating into the app where we can then download it and test it live.

Offline Client/Server Simulation:

On client request we created an offline simulation of our automated validator so that we could test functionality without worrying about integration concerns. This simulator allows us to assure quality in several ways. We can detect model weaknesses, model loadings bugs, and logic defects. The offline simulation works by "submitting" images to the "server" for validation and receiving various results and printouts in return. For example we can see specific model outputs for each image, the key/deciding model outputs, the overall submission label and confidence, and the determined state after confidence thresholding.

For example, two defects we spotted early on was the inability of the trashcan model to recognize dogs/animals and the same model giving dark/close up images a valid rating with high confidence. A round of retraining allowed us to correct both these issues leading to the results shown in figures 5, 6, and 7 below (the leftmost image is supposed to be the trash can)

```
The trash can image is most likely Valid with a 80.62 percent confidence.
The litter images are most likely Valid with a 92.62 percent confidence.
Label: Valid
Confidence: 80.61860799789429
State: Valid
```

Valid Submission - Fig 5

```
The trash can image is most likely Invalid with a 92.58 percent confidence.
The litter images are most likely Invalid with a 99.12 percent confidence.
Label: Invalid
Confidence: 92.57673025131226
State: Invalid
```

Invalid Submission - Fig 6

```
The trash can image is most likely Valid with a 53.73 percent confidence.
The litter images are most likely Invalid with a 69.66 percent confidence.
Label: Invalid
Confidence: 69.65987086296082
State: Unknown
```

Invalid Submission - Fig 7

Results

Unfinished Features:

When it came to what we couldn't complete, the unfinished portion of our project was constrained to only stretch goals. One such stretch goal was integrating our machine learning model into the Firebase platform via Firebase cloud functions, the Firebase realtime database, and cloud storage. This involved using the Tensorflow.js API along with Google's APIs to handle database and cloud storage read/write operations, and feed those operations into our machine learning model. Unfortunately, we had trouble with using just about every aspect of each API, which led to integration taking far longer than we had anticipated. Another incomplete aspect, which wasn't a stretch goal but rather an extra feature we proposed, was the trustworthiness score to limit the data needing to be fed into the model. We couldn't accomplish this because it was something to integrate after integration was working properly. Finally, we were unable to perform read and write operations to the database in the context of the trustworthiness score, as writing the score to the database would require it to be implemented, which did not come to fruition.

Performance Testing:

Because the client wanted us to create a cloud-hosted application, it was important to make the solution lightweight in addition to being effective. As such, we did some testing on our model's accuracy, confidence, and some performance testing in the form of analytics on CPU and memory usage. The client gave us a goal of 80% accuracy for our model, so we shot for that figure based on our model's accuracy (we achieved that value). As for confidence, this is a parameter that the client would like to control such that only classifications with a certain confidence can be accepted or denied. We used our best judgement here to make sure our model was making correct classifications with a high level of confidence. When it came to our CPU and memory usage, we made an effort to keep consumption as low as possible. We tested performance after every model modification, and if resource consumption went up, we would discuss how it could be improved, or if we needed to take another approach.

Testing Summary and Usability Tests:

Because our testing was all performance-based and integration wasn't completed, these concepts do not apply to our project.

Lessons Learned:

Through this project, the team learned a multitude of skills and lessons. For one, we gained a better understanding of Tensorflow concepts. Those of us who worked on the model got to better understand how to train a Tensorflow SavedModel in Python, while those of us working on the Firebase integration got to learn more about Tensorflow.js and how to run images through models with it. We also learned how to use metrics with Tensorflow; getting information such as confidence intervals, confusion matrices, and accuracy was important for the project. We also learned a great deal about project permissions, specifically with deploying cloud functions. We now know how to deal with authentication in this regard. This project also helped us build our soft skills, working with the Agile process and using Scrum. We learned how to properly communicate with the client to not only create our solution, but also to understand the problem that needed to be solved.

Future Work

Full Integration with App:

The next step is full integration with the Litter CleanUp app. Either use the foundation we created or use new code to integrate the machine learning models we provided into the existing app and Firebase databases. Alternatively, switch platforms to use existing and partially functional Python client-server script.

Further Improvements on Models:

The major way to improve the models at this point is to gather larger training sets and use them to retrain the models. Particularly, our trash can model is lacking valid images in its training set.

Another way to improve the models would be to implement more functionality that allowed the validator to detect untrustworthy activities such as taking 100 pictures of the same piece of litter to get more rewards. The models would have to be expanded to analyze the submission as a whole rather than just a single image at a time.

Trustworthiness Score Implementation:

Lowers performance cost as most of the submissions are provided by the same users and smaller sampling of those submissions would only increase efficiency. Have user submissions be randomly sampled based on an added user trustworthiness score to decrease the number of images that have to be processed through the machine learning models.

Automatic Feedback System:

Automated feedback to users about invalid/unknown submissions and reasons for that classification. For example, showing specific images that had problems running with the models. This was a new idea introduced near the end of our work so it might even be considered a separate project.

Appendices

- Development environment description
    - Google Colab for shared Python scripts
        - Model training and server-client simulation
    - Microsoft Visual Studio for *.js scripts
        - Cloud functions for Firebase integration
- Coding conventions
    - For our cloud function code we used ESLint, a linter and validator, to ensure our code followed ECMAScript 6 coding standards.
    - Code comments for order of how to run Python notebooks and what each does