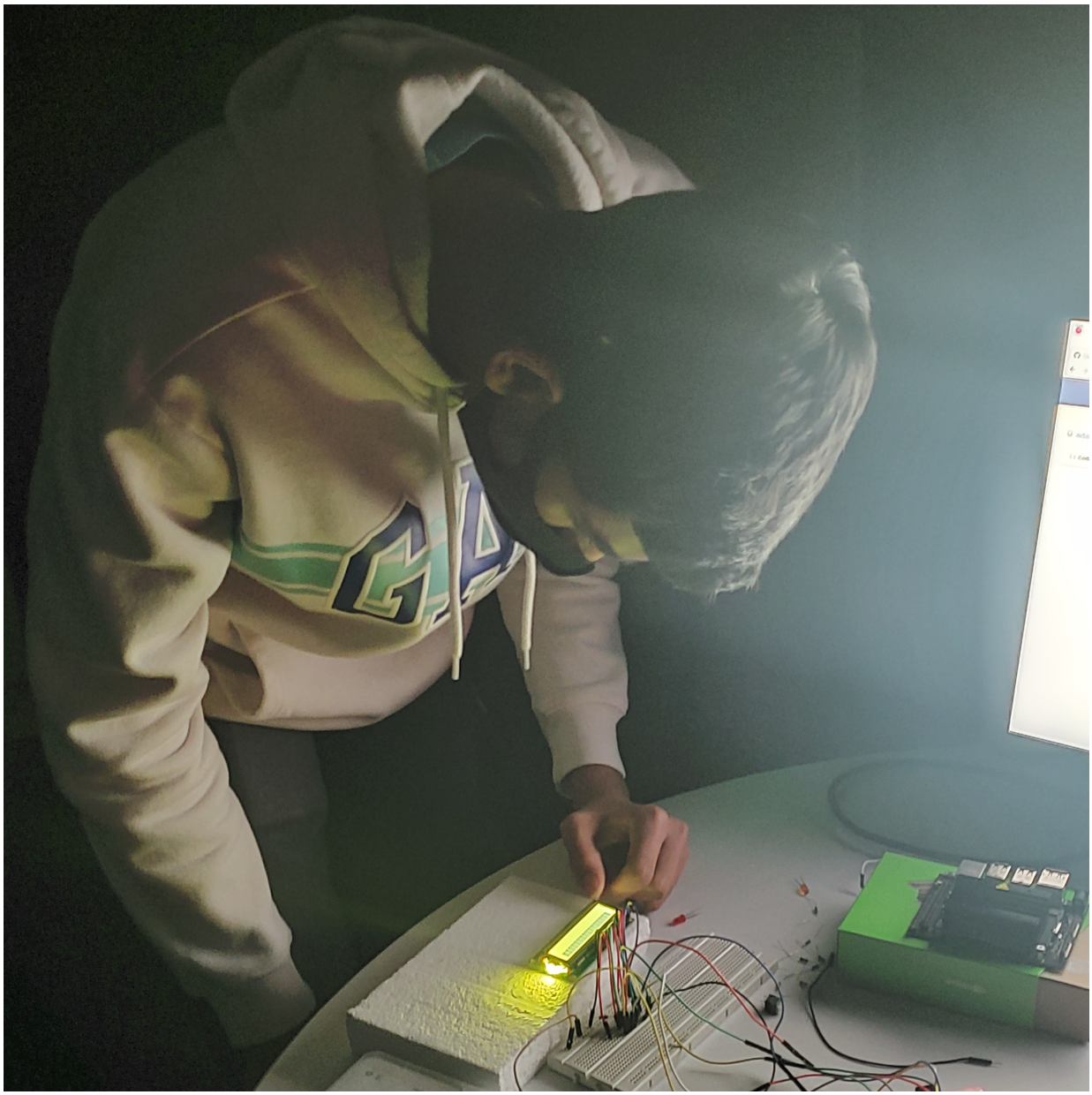


Project Trash-Classifier

10.28.2020 - 04.09.2021

TABLE OF CONTENTS

- Background Overview
- Project Goal
- Resources
  - The Team
  - Material Resources
- Execution
- Engineer's Review
- Timeline
- Fritzing Diagrams
- Notes
- Statistics / Benchmarks
- Conclusion
- Sources
- Links

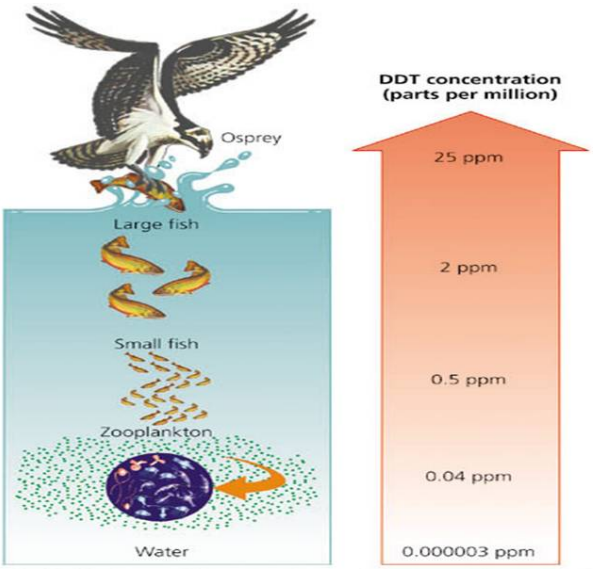
Background Overview

With increasing amounts of pollution, the amount of untracked waste is increasing rapidly. With acres upon acres of garbage condensing in the ocean, garbage "islands" like the Great Pacific Garbage Patch pose a significant, artificial threat to marine and intertidal species' way of living.

The GPGP spans twice the size of texas, frequently washing up on beaches (Schoell, Mary)

Excess, unmanaged waste is terrible for the environment **and** for humans. Landfills are swarmed daily by Racoons, Bears, and Deer looking for a source of food - In the end, all 3 of these species, and thousands of others, find harmful plastic and chemicals. In marine terms, this is also substantially pronounced - Fish, whales, and sharks looking for an easy food source eat plastic, harming themselves with no nutritional benefits.

Humans, too, are affected by this trash accumulation through something Ecologists call "Biomagnification." Microplastics at the bottom of the food chain make their way to the top, accumulating so that the top of the food chain is exposed to a larger concentration of plastics than the bottom. Humans, the top of this food chain, could become exposed to the highest plastic levels - A consequence most would prefer not to think about.



While pollution has been occurring even before the 1st Industrial Revolution, the past two decades have seen an alarming increase in pollution due to massive population growth, lack of dynamic restrictions, and a substantial dearth of public awareness. With less than 600 million tons of plastic being recycled in the US (out of 6.3 billion tons total), the amount of trash piling up in landfills can undeniably be substantially reduced (Downs and Acevedo).

In many developing nations, excess trash is burned - the main contributor to declining air quality. 15% of all Waste is dangerous in and of itself, classified as Radioactive, Infectious, Toxic, or a Biohazard (WHO). It can no longer be denied that mankind's impact on the environment is world-breaking, but the general population is still blissfully unaware and rarely take steps to remediate the problem.

Project Goal

To provide a solution to the lack of public awareness, our project aims to create industrial-grade solutions for the general public and trash management companies. Classifying different kinds of trash is something that confuses many people, so our goal is to create a solution that identifies and classifies trash with a focus on ease of accessibility. Furthermore, for industrial applications, we intend to develop a hardware form of our solution that can classify trash, aiding in the actual trash Sorting.

In terms of set goals:

1. On-device image inference on multiple hardware platforms - Consumer Use
2. Offloaded image inference (Cloud), logging, and basic alerting - Industrial Use

Resources

The Team

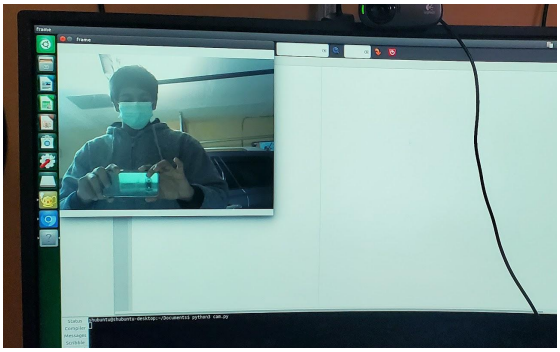
A group of enthusiastic future engineers, our team seeks professionalism wherever we go.

Aniket Bhosale

The hardware specialist of the team, Aniket Bhosale aids the team with his electrical engineering knowledge. With just a breadboard, a few cables, and some LEDs, he helps add significant functionality to our Raspberry Pi's image classification.

Govind Gnanakumar

The team's all-rounder, Govind displays his expertise in many fields by training and testing the model, conceptualizing the architecture, providing helpful advice to the team, and refactoring our solutions to be more efficient, more productive, and most of all, more professional.



Shynn Lawrence

The software whiz, Shynn is the Pythoneer of our team. Bringing a software developer's unique perspective, he helps the team run smoothly and helps iron out solutions to any of the problems that the team faces.

Signed: Shynn Lawrence, Aniket B, Govind G

Material Resources

Devices

- Raspberry Pi 3B+
- Raspi Cam v1
- Jetson Nano (4 GB Memory)
- Generic Powerful Desktop
    - More information in Engineering Notes
- Generic Laptop
    - Android Application Development
- Samsung S10e and S21 (Mobile App Testing)

Hardware Tools

- Generic Breadboard
    - Pictures in Engineering Notes
- Generic Resistors
- 4 Separately colored LEDs
- GPIO cables

Software Tools/Libraries

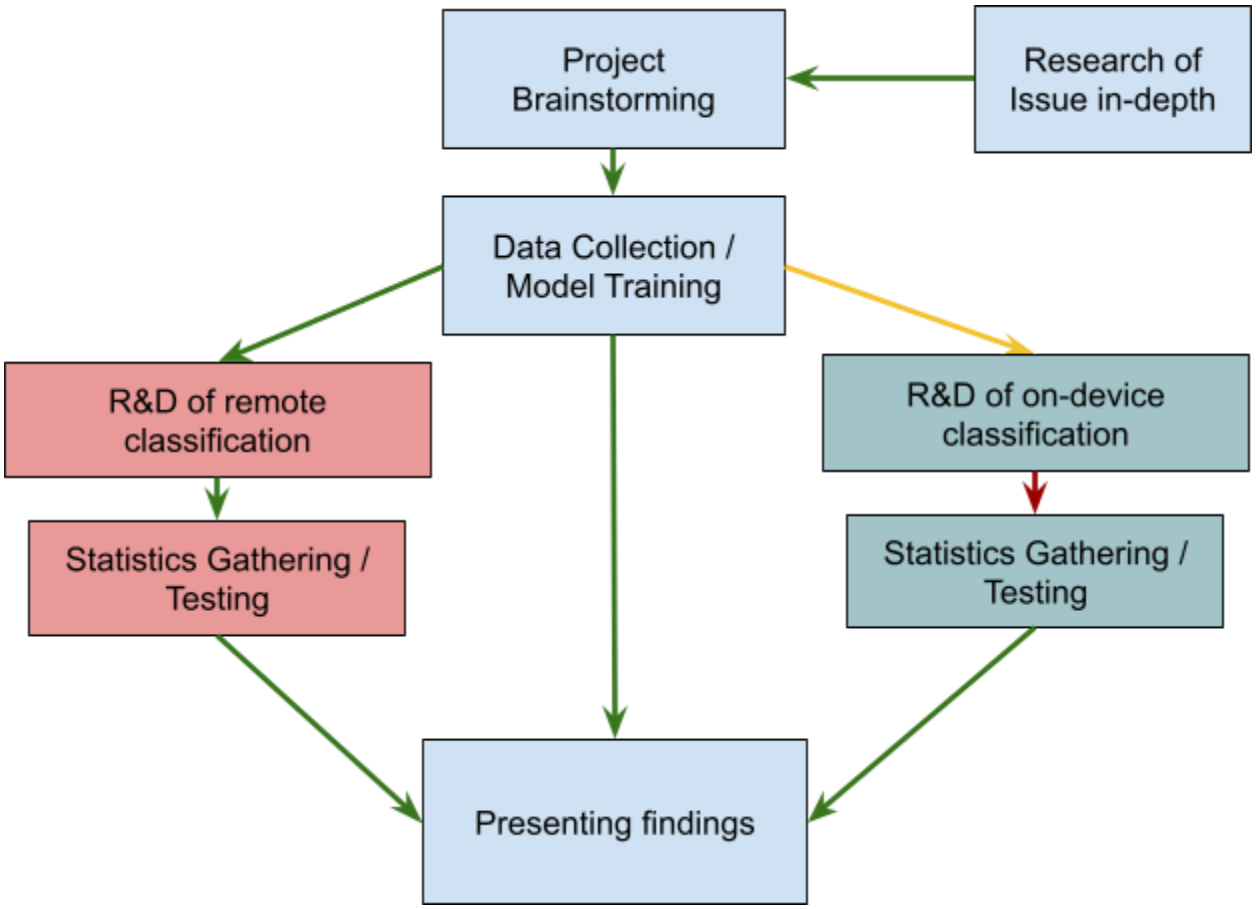
- Python 3 (Standard Library)
    - Our source code can be found in Engineering Notes
- Lobe
- Tensorflow 2
- OpenCV
- Pillow
- Android Studio
    - Kotlin on JVM for development

See Links to Tools for more information on these tools

Execution

Development Plan

Our two project goals work on shared resources, and as such, we followed this Development Plan:



Signed: Shynn Lawrence, Aniket B, Govind G

Engineer's Review

Hello Shynn, Aniket, Govind,

I'm very happy to be able to provide a review for your "Trash Classifier" project.

A brief background on myself. I'm an engineer with 20 years of experience working at Intel (12 years), contributing to the development of upcoming generations of Intel GPU architectures, and with prior experience including past CPU & System architectures along with building optimizing compilers for the same. Informally speaking, it involves contributing to the development of Intel GPU/CPUs to execute Deep-Learning training & inference frameworks, as leveraged in your project, among others, with high performance & efficiency.

Now for my project review. In my opinion, you have wisely chosen a very useful project with potential utility for productization, and with the right level of complexity for a good high-school deep learning project. I can certainly imagine your vision of this proof-of-concept implementation being productized and leveraged as part of a larger solution for full or partial automation in the trash recycling industry, as well as for more ubiquitous use for the average person via an Android/iPhone app, with the latest phones being able to run inference sufficiently fast using specialized hardware accelerators for the same.

Your planning for the project appears to be well thought out. The design choice of using the Raspberry Pi platform for the initial implementation with the TensorFlow Lite model for classification sounds like a reasonable and adequate choice, and the same can be said for using the python Keras API with TensorFlow 2 platform to create the trained model. Validating that the inference obtained good enough results, i.e., accuracy, with the trained model, per the data set you gathered, was the right choice for a first milestone.

On encountering the less than hoped for inference performance with the Raspberry Pi platform, being able to quickly port and test your inference on the higher-powered Jetson Nano platform to achieve 4x better performance was very impressive. You were

Signed: Shynn Lawrence, Aniket B, Govind G

quickly able to recalibrate efforts and projections for performance targets and thus update the upcoming milestones on the schedule to meet deadlines with high confidence while being able to allocate time for the remaining less risky porting effort for camera API and GPIO programming for the control system programming on the output LEDs. This was indeed a good demonstration of problem-solving in overcoming roadblocks encountered, as well as project execution.

Your stated plans for an Android app for the same and assessing performance on Samsung S10e is headed in the direction of a first productization of the proof-of-concept implementation (where even 1 fps is sufficient). The idea for you to potentially leverage cloud-based inference, e.g. AWS, to explore much higher performance for video (~24 fps), is something to access to explore potential project configurations and their performance targets.

In summary, I'm impressed with your project choice, it's potential utility, planning, scheduling, improvisation, execution, and the clear presentation of your ideas. Looking forward to hearing more about the progress made.

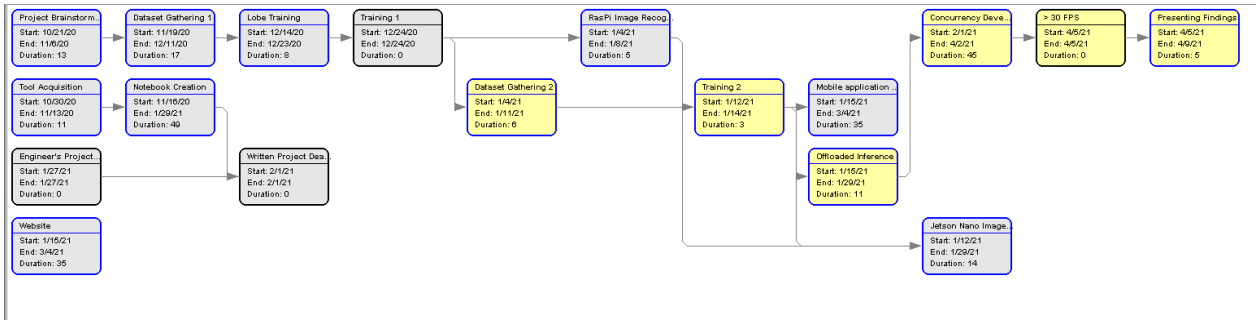
Regards,

Biju George (biju.george@intel.com)

Timeline (Gantt, PERT)

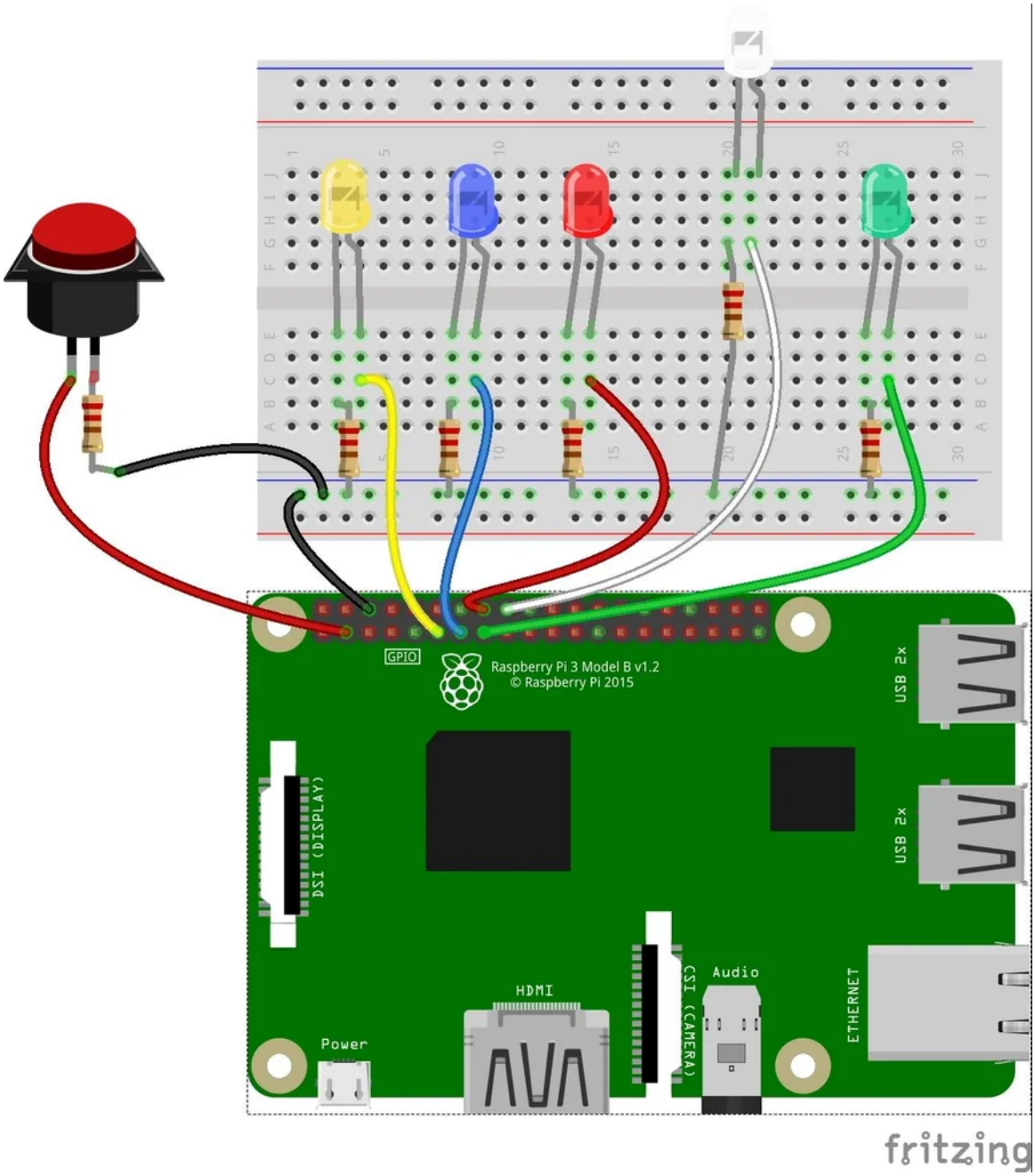
Click to view our Gantt Chart.

Click the PERT chart's hyperlink to view it in more detail:

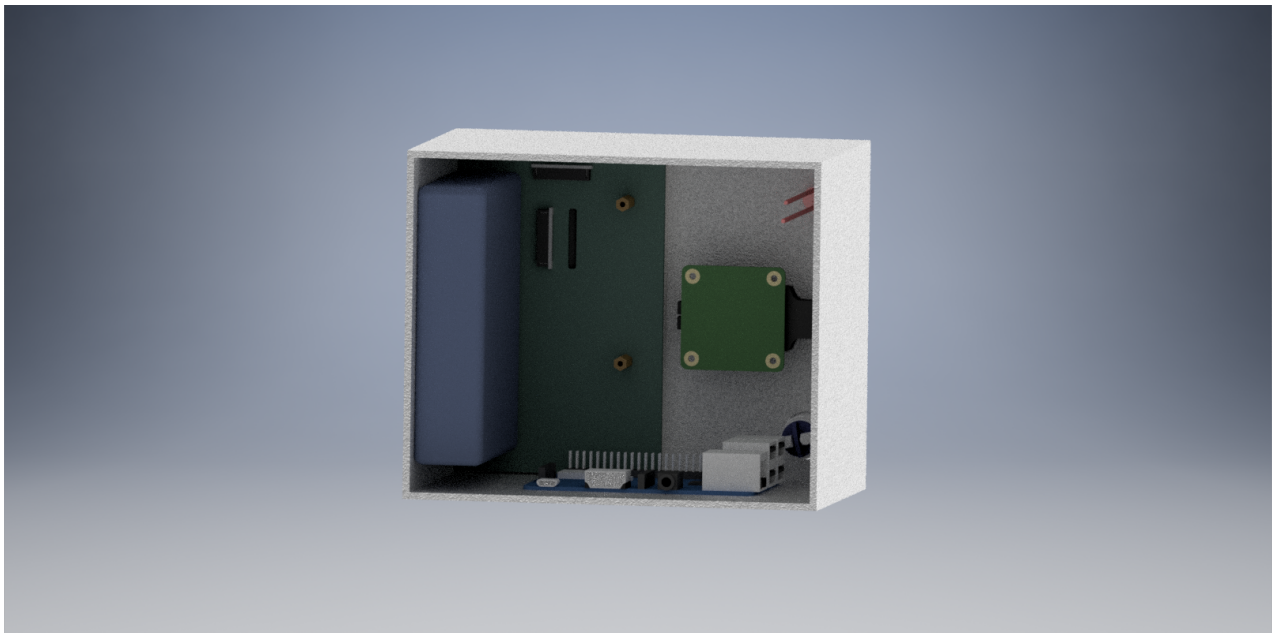


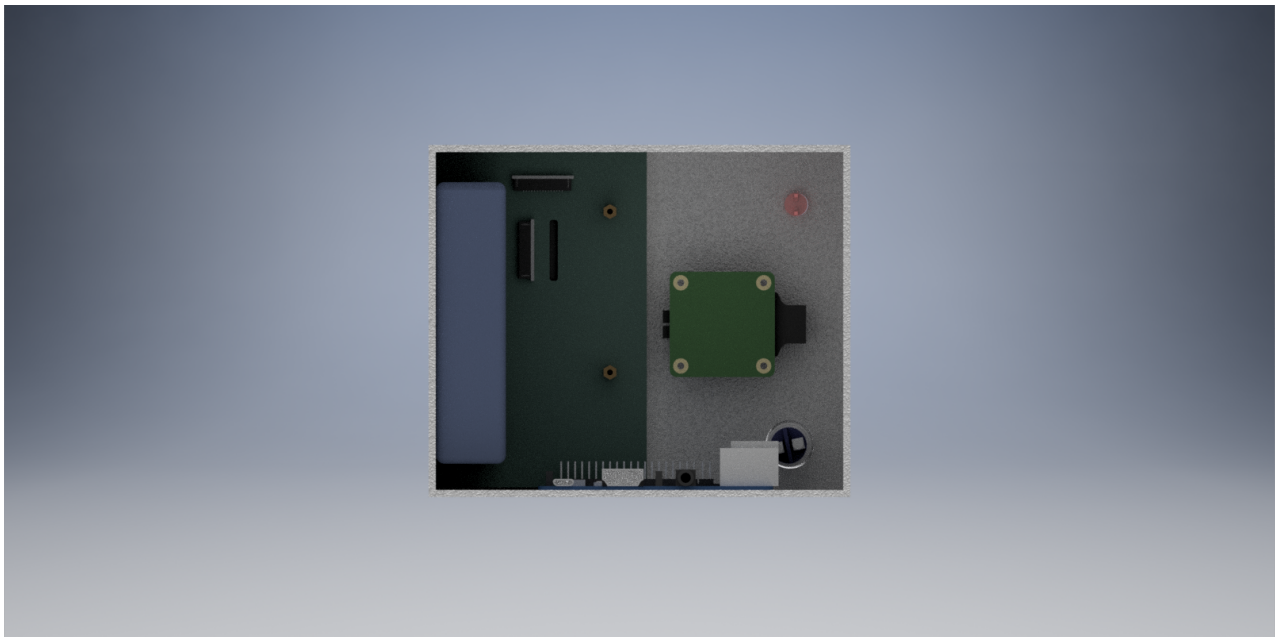
Signed: Shynn Lawrence, Aniket B, Govind G

Fritzing Diagrams



Signed: Shynn Lawrence, Aniket B, Govind G





Notes

The next few pages are dedicated to detailed notes of our engineering process, how we achieved our performance, and the issues we faced (particularly how we overcame them).

Signed: Shynn Lawrence, Aniket B, Govind G

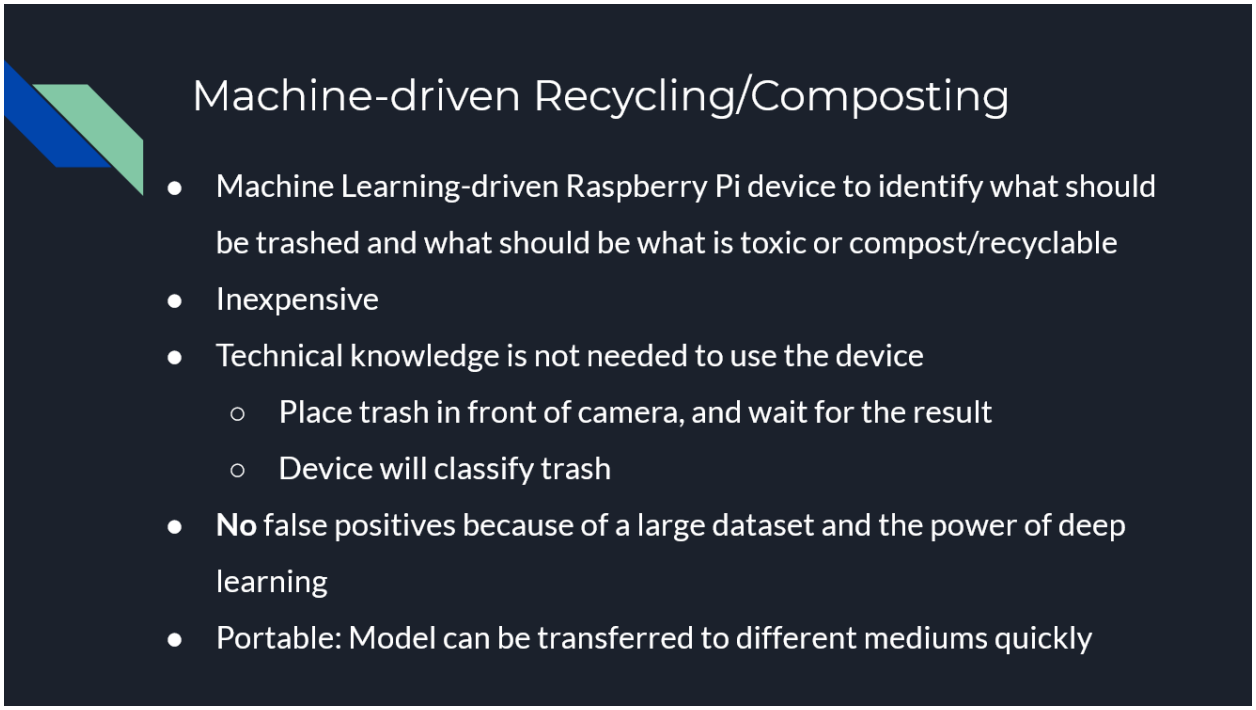
10/27/2021 - Project Idea

Today the team brainstormed for the SkillsUSA Engineering/Design project. Analyzing global issues, different ideas (many not even close to being feasible) were thrown around. There are thousands of issues that affect humankind and Earth: Global Warming, Overpopulation, World Hunger, etc, but we chose an issue we found affected the most people - mass pollution. Many developing countries worldwide burn their garbage, and even at home in the US, our waste is either shipped to other nations or piled into landfills. Governments are unwilling to take these matters into their own hands, allowing the trash to keep on proliferating. After unanimous consent, we decided to pursue a solution to that end. We began creating the first few slides in a slideshow, highlighting the problems of trash proliferation and the lack of public awareness.

Signed: Shynn Lawrence, Aniket B, Govind G

11/03/2020 - Solution Ideas

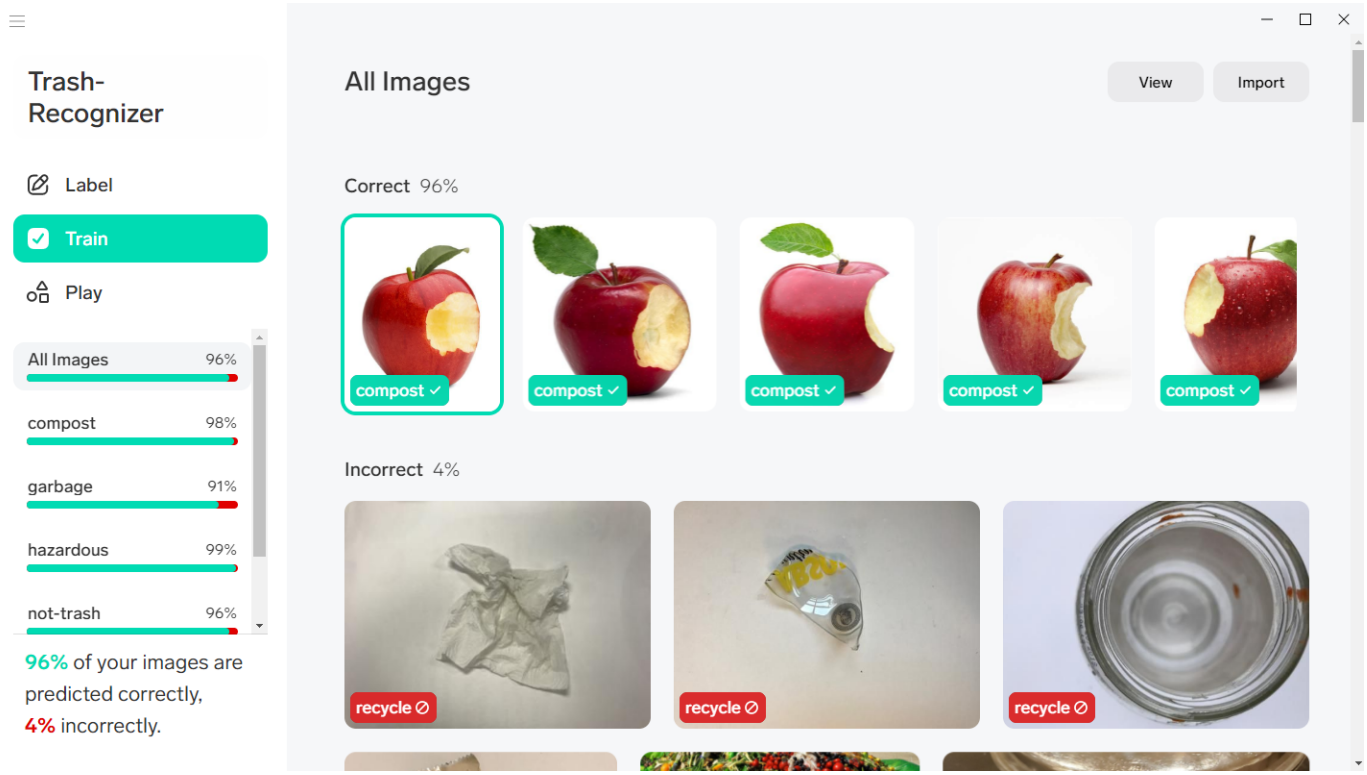
After identifying the problem we want to solve, we are beginning to find solutions. The prime solution to the problem, something we are interested in pursuing, is using AI to automatically classify trash. AI is an enticing prospect because it is a completely mechanical solution to the problem - Devices can run 24/7, humans cannot. Further research is required, but we have completed our slideshow highlighting the benefits of using Machine Learning to classify trash, and we'll start gathering a Dataset soon.



Signed: Shynn Lawrence, Aniket B, Govind G

12/14/2020 - Beginning Training

Dataset Collection has essentially been completed, and we aim to search for ways to train a model using our dataset. We found a tool (Lobe) that will help automate the training process for us, which is something we're interested in pursuing. Govind will be using Lobe to train the model and then export it in the TensorFlow lite form factor. Tensorflow Lite is perfect for consumer-grade on-device inference because it allows faster inferences, a smaller model size, and higher energy efficiency.



Signed: Shynn Lawrence, Aniket B, Govind G

12/24/2021 - Training completion

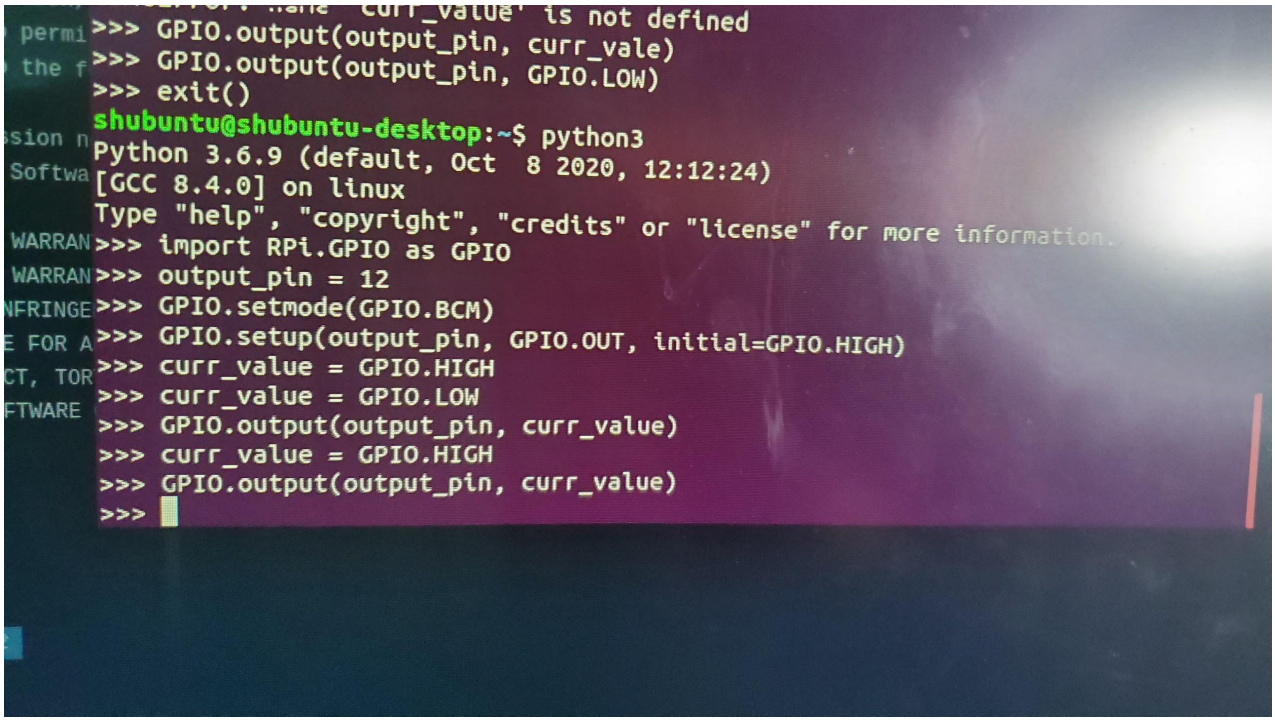
Milestone - Basic Training has been completed, and the tensorflow lite model has been exported.

Name	Date modified	Type	Size
example	1/16/2021 11:16 PM	File folder	
saved_model.tflite	1/26/2021 8:20 PM	TFLITE File	102,145 KB
signature.json	1/26/2021 8:20 PM	JSON Source File	1 KB

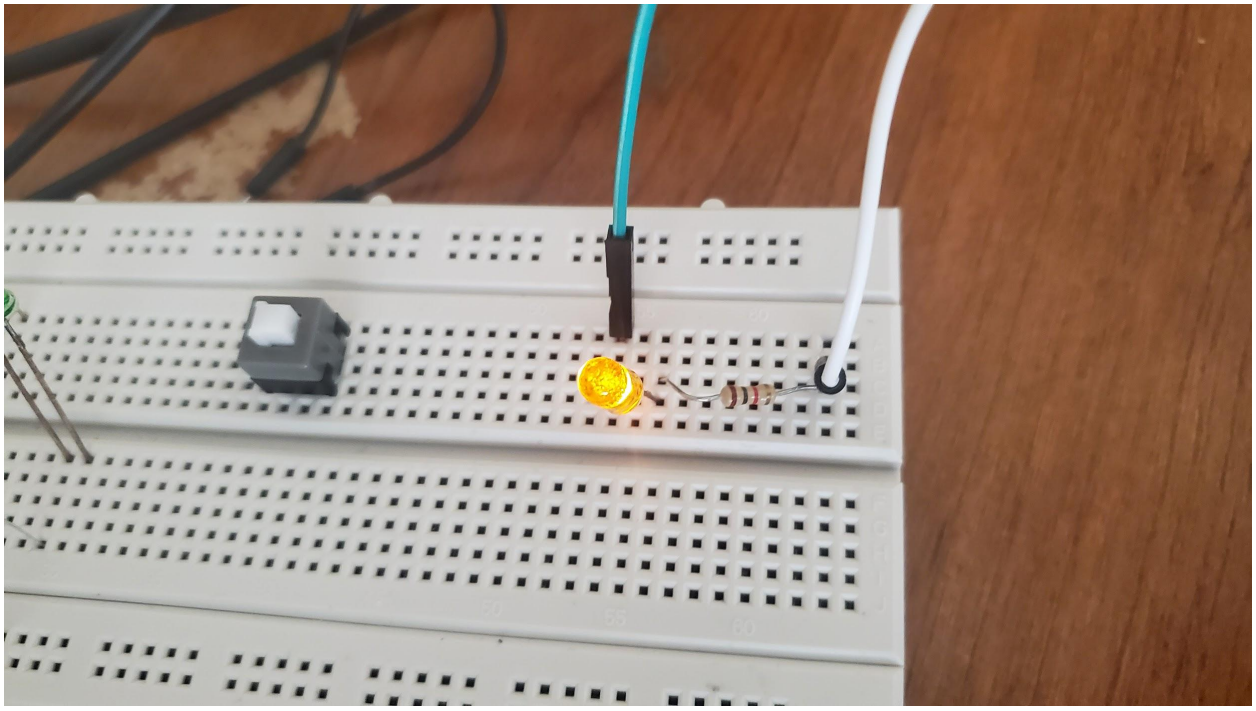
The resulting model is a little over 100 Megabytes, a bit large but the seemingly high accuracy makes it a small price to pay.

1/4/2-21 - GPIO Development, Engineering Contact

After a short Winter break, progress in the project is continuing to be made. Raspberry Pi testing begins, and we're working on developing the raspberry pi's GPIO LEDs and testing the inference on the Raspberry Pi (on-device). We've also contacted a local software engineer (Systems Architect), Mr. Biju George of Intel Corporation, to discuss our project with him and receive his feedback on our project.



After an hour, we were able to get the GPIO pins to turn on or off at will



The Orange Light in the 1 (On/GPIO.HIGH) state

Signed: Shynn Lawrence, Aniket B, Govind G

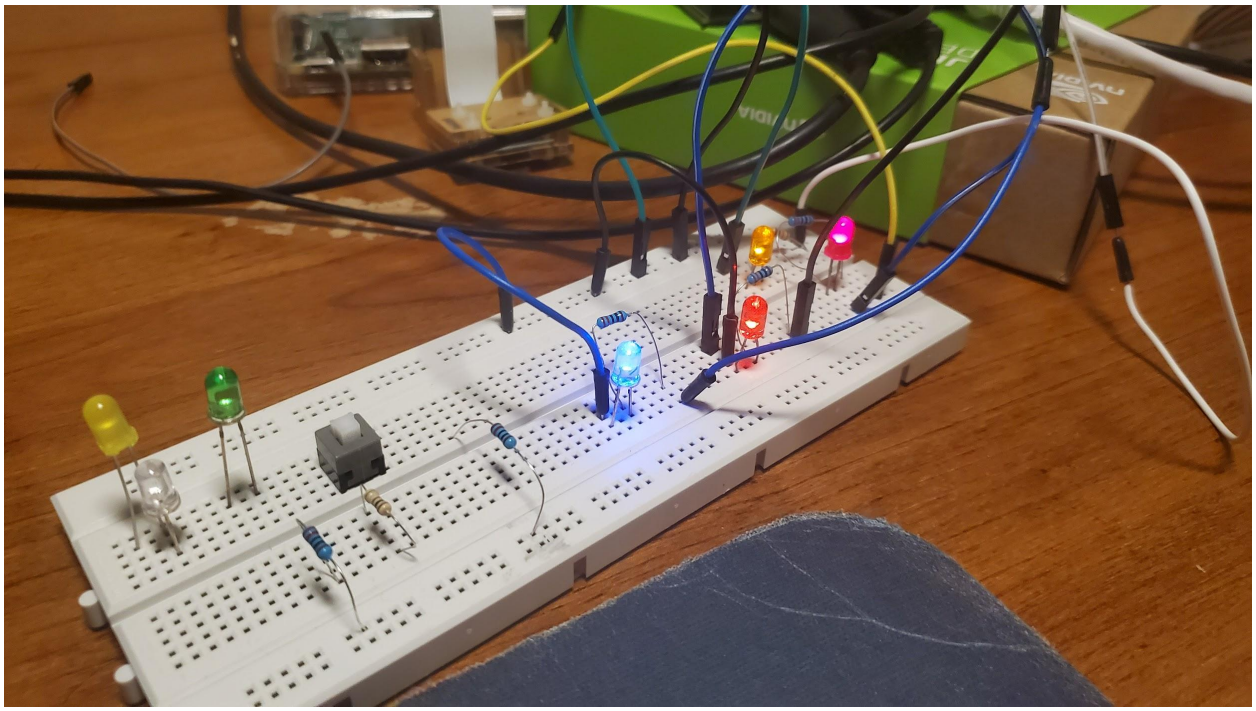
1/12/2021 - Poor Performance, Further Testing

After actually testing the inference on the Raspberry Pi, we got less than admirable results. We have noted these results on the slideshow, which we intend to show to the judges on the day of the competition. In short, the Raspberry Pi's frame rate was around 0.25 fps (~1 frame per **4** seconds), something that, while not completely unexpected, convinced us that the raspberry pi should not be used as a demonstration of on-device performance. To this end, we intend to hook up the GPIO to a 4 Gigabyte Jetson Nano, another minicomputer with a higher price but better performance and test out the performance of the model.

1/17/2021 - Jetson Problems, Continued Expansion

The Jetson Nano unit that we have has a hardware deficiency in that it does not boot up again after it is shut down. Unfortunately, after hours of searching, there doesn't seem to be a viable solution to this problem. We can still test the device's performance, just cannot shut it down... Preliminary tests show that the device squeezes out somewhere around 1fps, significantly better than the Raspberry Pi's performance. We intend to show the performance to the judges on the slideshow, just like before.

Signed: Shynn Lawrence, Aniket B, Govind G



While the Jetson Nano itself has hardware issues, the GPIO works perfectly

Govind has started the development of an Android Application and a website, both of which fit in perfectly with the part of the project that gears towards general consumer use. Aniket and Shynn are shifting gears to offloaded inference through sockets - Sending images to a powerful computer, inferring the image, and sending the results back to the client - a process that emulates the enterprise usage of cloud computing for faster speeds.

MainActivity.kt × build.gradle (:app) ×

Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properl

```
}
buildscript {
    repositories {
        maven { url 'https://google.bintray.com/tensorflow'}
    }
}

android {
    compileSdkVersion 30
    buildToolsVersion "30.0.3"

    defaultConfig {
        applicationId "com.example.trashclassifier"
        minSdkVersion 16
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
```

The current Gradle Task for our Android Application

1/24/2021 - 2nd Training deployment, Awaiting Review

A few days earlier, training on a larger (~2x) dataset was completed and now we are "rolling out" the changes to our current device set. The new model is the same size but should offer significantly more accurate real-world results, especially in the areas of compost and non-trash. A basic client-server python program has been set up, to be continued in the offloading part of the project. It will eventually be connected to the GPIO program and the Inference Program to be demonstrated to the judges.

```python
import socket
import cv2
import pickle

img = cv2.imread("someImage.jpg")

s = socket.socket()
print("Socket successfully created")

port = 12345

s.bind(('', port))
print("socket binded to %s" % (port))

s.listen(5)
print("socket is listening")

c, addr = s.accept()
print('Got connection from', addr)

while True:
    dumped = pickle.dumps(img)
    c.send(dumped)
    a = c.recv(2_000_000)

c.close()
```

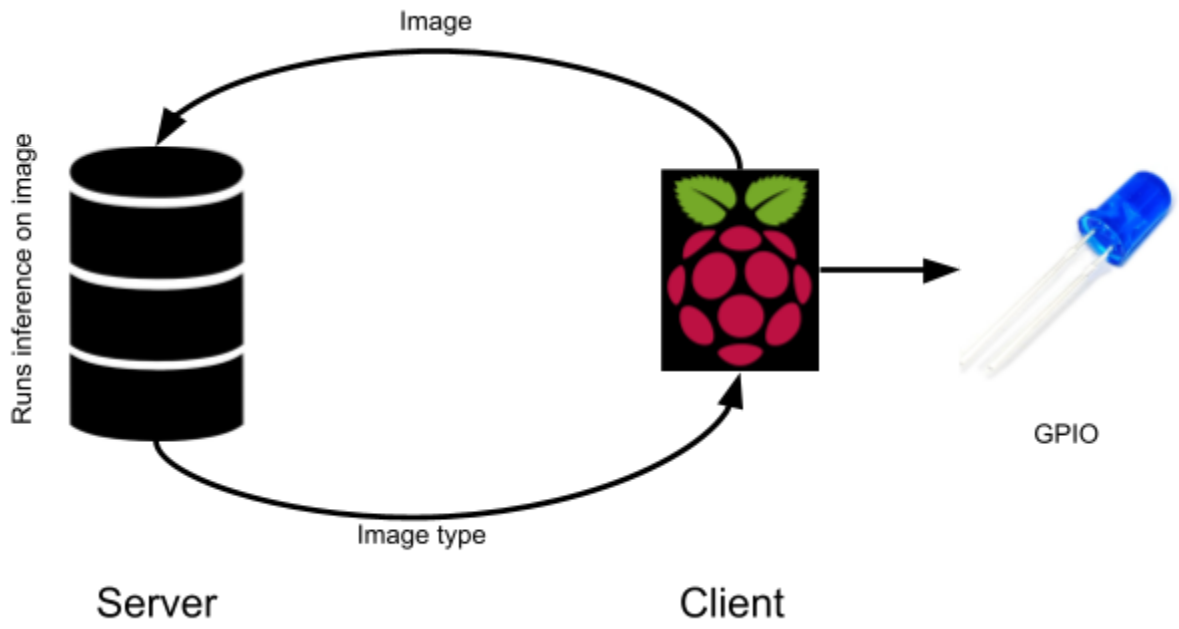
Server.py

Signed: Shynn Lawrence, Aniket B, Govind G

```python
import socket
import pickle
import cv2

s = socket.socket()

port = 12345

s.connect(('127.0.0.1', port))

while True:
    receivedObject = s.recv(2_000_000)
    print(len(receivedObject))
    unpickledObject = pickle.loads(receivedObject)
    cv2.imshow("Received image", unpickledObject)
    cv2.waitKey(0)
    s.send("Received the Image".encode("Utf-8"))

s.close()
```

Client.py

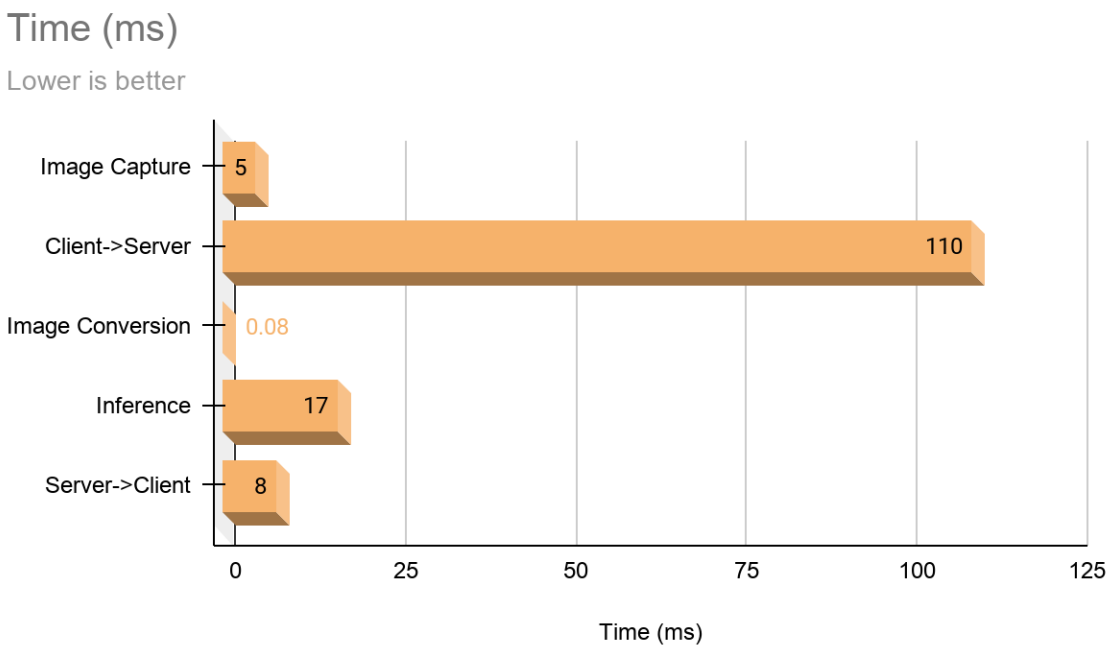
The Client.py Script would run on the Raspberry Pi device, using the PiCam v1 to take images. The socket, which would be established between the Raspberry Pi and a "server" (just a home pc). At that point, the Raspberry Pi would send every image it takes to the Server (through the network). The Server would then run the inference script on the received image, interpret the result, and send the Class (the label) back to the Raspberry Pi. The Raspberry Pi converts the label to its corresponding LED Light (Dictionary, Label -> GPIO Number) and enacts the GPIO script. The GPIO script, the last step of the process, will send a signal through the GPIO pins to display the specific light, and 1 cycle of the offloaded inference is completed. A helpful diagram is shown below, to highlight exactly what's going on:



The source code shown above is not ready, and the server-client code is flipped - It is only for testing the transfer of objects. The full synchronous code can be found on this git gist. A Video Demo of the synchronous code can be viewed by clicking on this link.

2/20/2021 - Using Concurrency to increase efficiency

An engineer overseeing our project, Mr. Biju George (See Engineering Review above), suggested that instead of using a fully synchronous system, concurrency could be used to speed up the off-loading architecture significantly. For reference, here is the performance breakdown of the current synchronous architecture:



This total synchronous architecture's time adds up to an estimated 140 milliseconds per image inference (including the time spent on sending the result back to the raspberry pi). In other words, an average of 7 frames per second. The current goal is to boost this up as much as possible before the next competition. As Mr. George noted, most films are recorded at 24 frames per second, so achieving such a level would be a good milestone, because it gives a quantifiable and achievable goal. Currently, 3 different Python libraries are being considered:

- Asyncio - A python module allowing asynchronous development through the famous async/await syntax.
- Threading - A library allowing for the creation of "Threads", the Threading library is possibly the most popular asynchronous tool for beginners, it gives a glimpse into the methodology of concurrency but doesn't deal with issues like global variables being written at the same time (though it does provide tools to do so).
- Multiprocessing - The multithreading library does support concurrent programming, but because of Python's GIL (Global Interpreter Lock), the program running concurrently only seems so (it is actually merely using task-switching). The Multiprocessing module gets around this by spawning a new Process and running the python code in that new Process - This means it is limited by core count but is truly concurrent.

3/13/2021 - Development continues

The Multiprocessing module was chosen for its in-depth documentation. Because the Raspberry Pi (the “Client”) only has 4 physical CPU cores, only 4 Python Processes can be running at once.

Concurrent Client Code

To make the framework concurrent, the cores were split unevenly into two categories: “frameworker” and “worker”. 1 Frameworker process would constantly snap images using the camera, and then share the images with the 3 worker processes, who would send the images. At the startup of each program, multiple things occur in sequence on the client-side:

1. A Multiprocessing Pipeline is established for the use of transferring data
2. A Multiprocessing Lock (aka Semaphore) is established so that the workers can receive the images from the pipeline and send the images, something that uses sharing resources (so a Lock is necessary lest there be memory corruption)
3. A TCP/IP socket connection is established with the Server (currently through port 5000)
4. Processes containing methods for the frameworker and worker are created, passing in the lock, pipeline, and socket as arguments
5. The Processes are started

Here are a few images outlining the process:

```
68    parent_conn, child_conn = mp.Pipe()
69    lock = mp.Lock()
70
71    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
72    port = 5000
73    address ('192.168.86.110', port) # IPAddr is local server's address
```

Line 68 - Step 1

Line 69 - Step 2

Line 71-73 - Steps 3-5

```
79    jobs.append(mp.Process(target=frameworker, args=(queue, lock, sock)))
80
81    for i in range(3):
82        work = mp.Process(target=worker, args=(queue, lock, sock))
83        jobs.append(work)
84
85    for i in jobs:
86        i.start()
```

Lines 79-83 - Step 4

Lines 85-86 - Step 5

Concurrent Server Code

The Server-Side code works almost directly opposite to the Client-side code - where the Client sends, the Server receives, etc... Once again, 4 separate Processes were used, 3 for receiving and unpickling (Converting from raw bytes to usable data) and a 4th for the actual inference (which is, as seen in the chart above, 17 milliseconds). While the Processes were set up well, and they worked perfectly as intended, the Server-Side Pipeline had some strange issues in which images “sent” (The method used to push into the stack structure) would be collided with other images, leading to unmanageable and corrupted data in the inference method. Development is at a standstill because of this confusing issue, and solutions ranging from wrapping the pipeline within the semaphore to step-by-step debugging to find the issue have all been to no avail. Regardless of this lack of development, the program is achieving a respectable 13 frames per second, without sending the inference back.

To see a snapshot of this commit, please visit the Commit on Github.

Note - Due to uncertainties and inexperience with the use of concurrency, the files are named “testClient” and “testServer” for the Client and Server code respectively. They will be changed once the program is certain to work with a high degree of success, such that it can reliably replace the old, synchronous, “client.py” and “server.py” code.

3/27/2021

While the use of a pipeline is traditional, after further thought, a limited concurrent Queue was decided as a better solution. The pipeline was truly a hazard in that it set no limits of how many Images would be in transfer at once, meaning that if there was a bottleneck anywhere throughout the client code, it would just eat up memory, filling up with images, until it ran out of memory - dangerous business indeed. A limited queue is a perfect mix of a pipeline and a limiter in that it limits the amount of data that can be stored within it but is still concurrent-safe (meaning multiple Processes can access it at once without data corruption, data races, etc). An initial size of 16 images in the Queue is set, and the Queue will prevent any method from adding images beyond this limit. For more info on the Multiprocessing Queue and its "Blocking" calls, see this link.

The Queues worked perfectly when implemented, but didn't give significant performance boosts. At this point is when it was realized that using Processes garnered no significant advantage because Processes are best for CPU-based tasks while sending an image is an io-based task.

To utilize the processes to their greatest extent, LZ4 compression was added. LZ4 is a fast compression library that increased our frame rate to 15 fps. This also gave the benefit of utilizing the Processes for CPU-bound tasks, as they are originally meant to be. As of now, sending the resulting inference (from the Server to the Raspberry Pi Client) isn't possible, as all 4 Processes are currently being used - The Client cannot concurrently receive the images, because it is using the max number of Processes already.

A solution to this problem - using multiple threads in conjunction with subprocesses - is already in the works, and is currently in testing. Further developments will be noted down in this journal as soon as they are made.

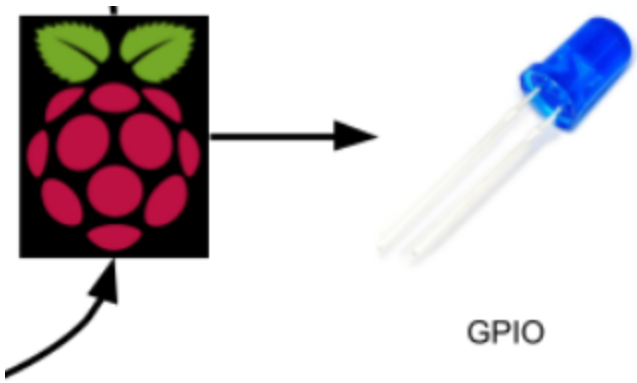
4/3/2021

The use of the Threading library in conjunction with Processes is a success, allowing many more things to occur at once and it subsequently solves many more of our problems at once. Many separate threads have been added to each process. A summary changelog -

- Client-side:
    - Each Worker Process spawns 2 threads, meaning instead of 3 images being compressed at the "same time", 6 are. Unfortunately, due to the limitations of resource-sharing in terms of the socket, this doesn't produce as visible of an effect as hoped - sending is still somewhat synchronous. Some further

compromises were made, however, they are negligible and end up helping the project more than they hurt it.

- The Frameworker Process spawns 3 separate threads, of which **2** are for image capturing. These 2 share a lock to capture and push to the shared Queue, meaning that while 2 images may not be captured at the same time, 2 can be added to the Queue at almost the same time. The 3rd thread, aptly named "display", receives the inferred image from the Server and displays it on the Raspberry Pi's GPIO pins. In other words, this is the fill-in for this part of the offloading architecture:



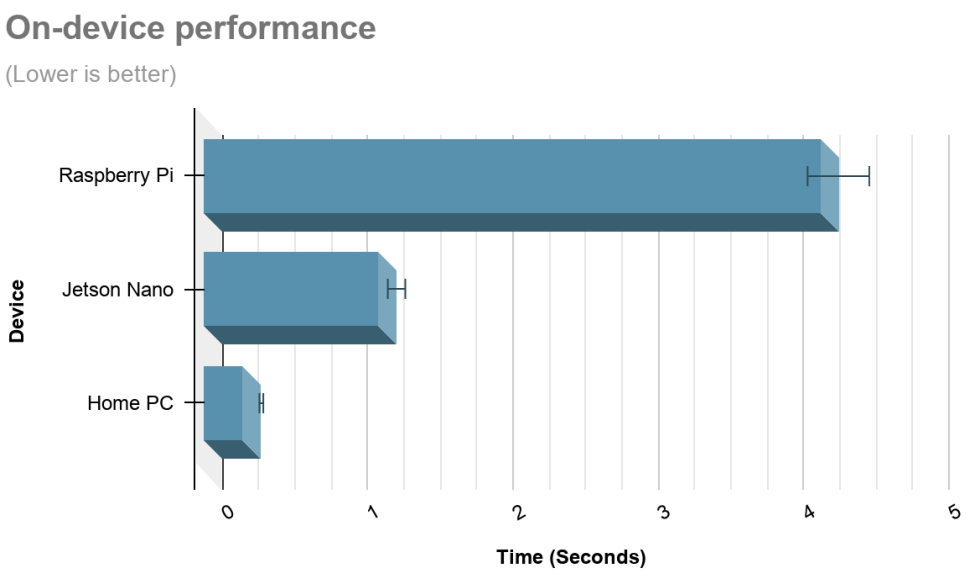
- A final, major change, is that the Raspberry Pi resizes the image to a 224x224 format, saving down the size of the image being transmitted along with being the exact size (Geometrically) that the model accepts (Model Metadata).

- Server-side:
  - Each Image-Receiving Process spawns 2 Threads, similar to the Client, meaning that 6 Threads are ready to receive images at the same time.
  - The Inference Process (named "inferAndSend" in the Program) now spawns 2 additional Threads, both for the purpose of sending results back to the Raspberry Pi. In addition, the original Process will continue in its duty of inferring images from the Queue and displaying them to the User (Testing purposes).

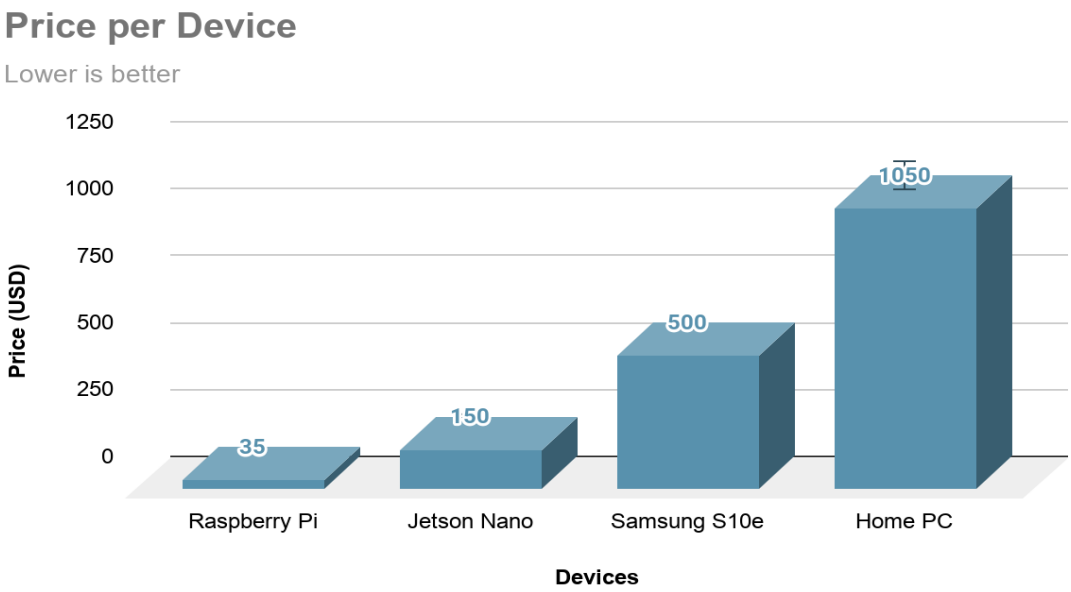
The use of concurrency and pre-resizing has yielded over 30 (surpassing our goal ~24 fps) frames per second, and that too without any compromises in terms of video smoothness. For more information on the change in inference frame rate over the course of our projects, see the Statistics section. The final trash-classifier remote code can be seen on Github.

Any other resources are available upon request.

Statistics / Benchmarks

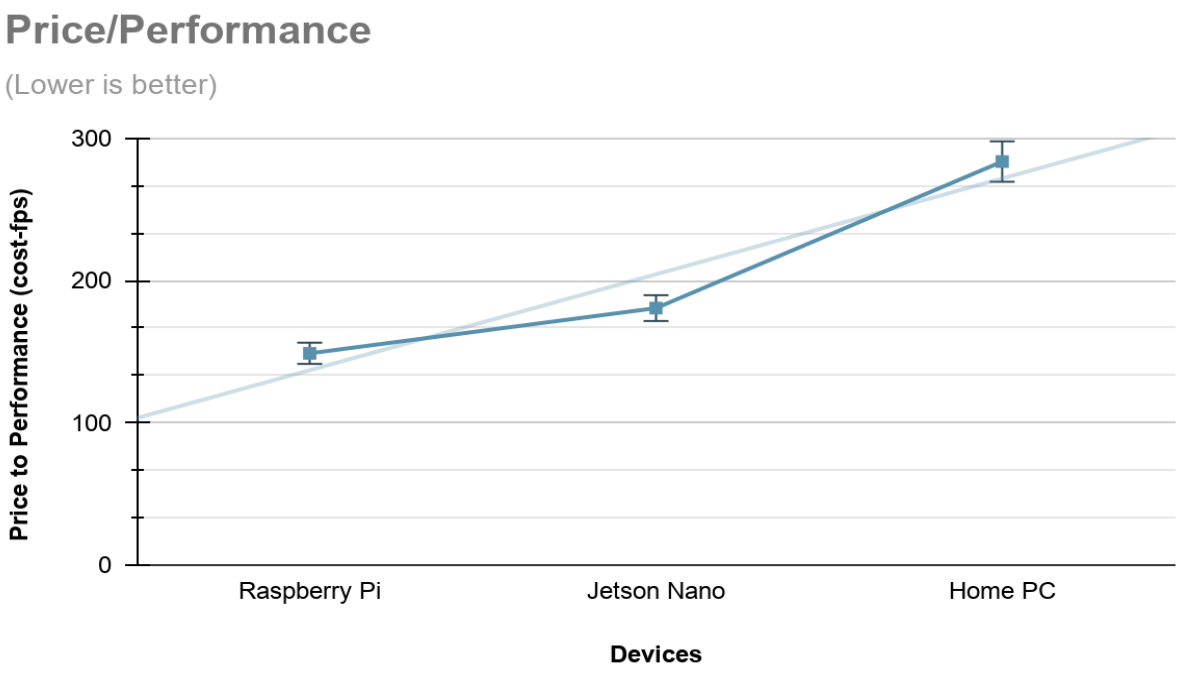


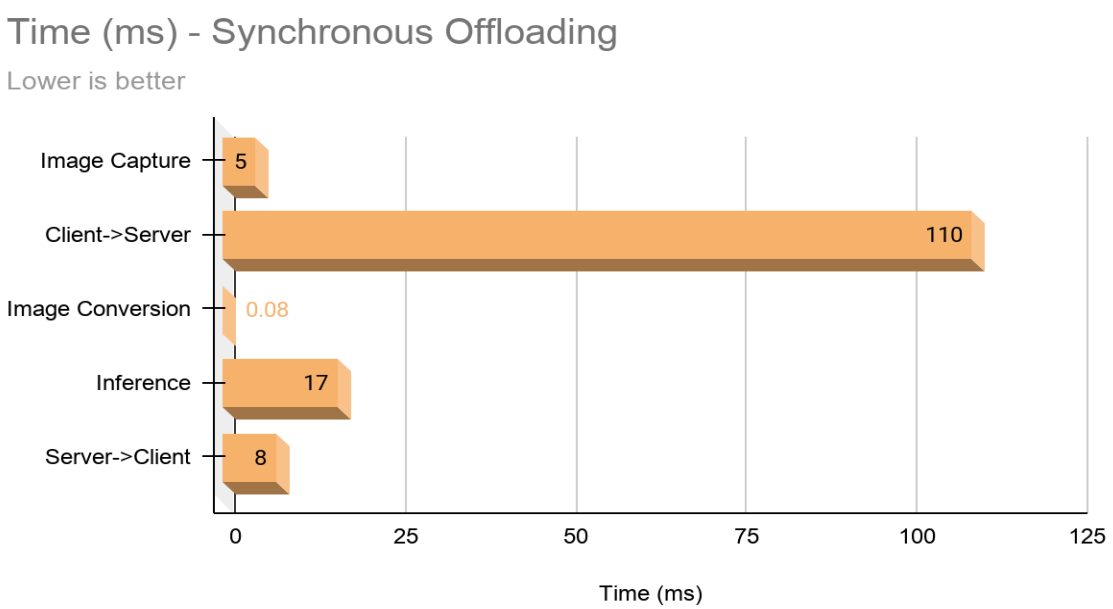
This analysis of the Device vs. inference time shows the comparisons between the different types of devices that were used in our testing stage.



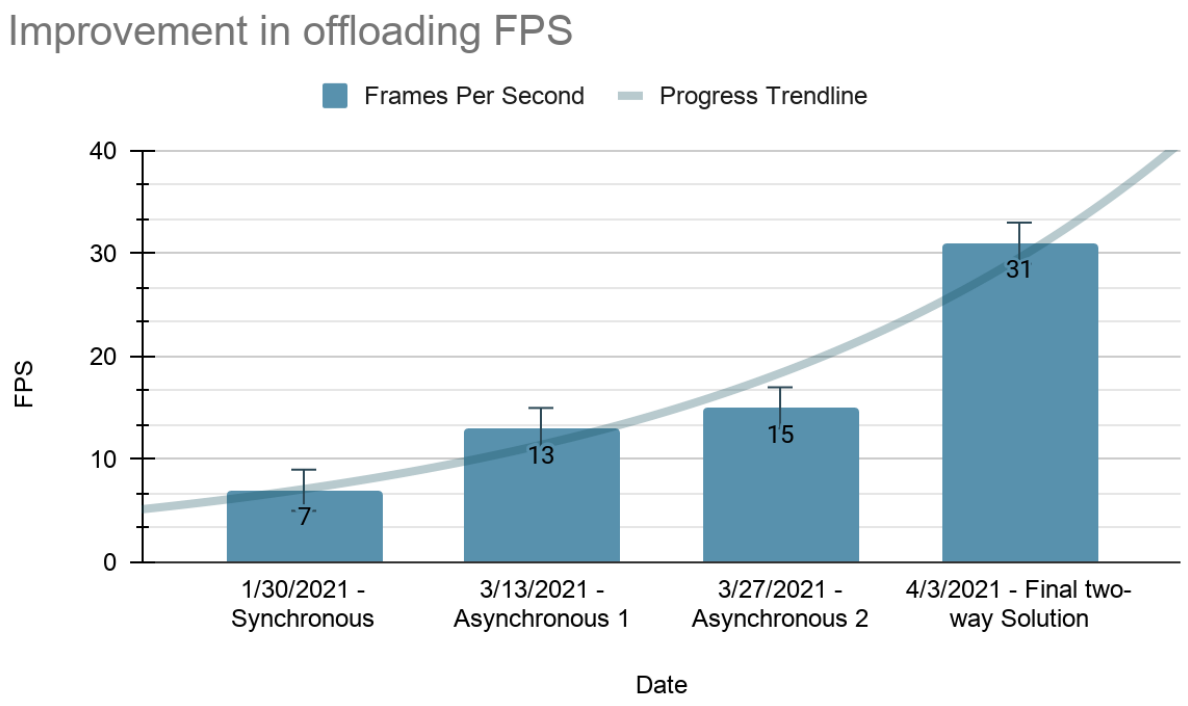
A cost-comparison chart used in our decision-making process.

Signed: Shynn Lawrence, Aniket B, Govind G



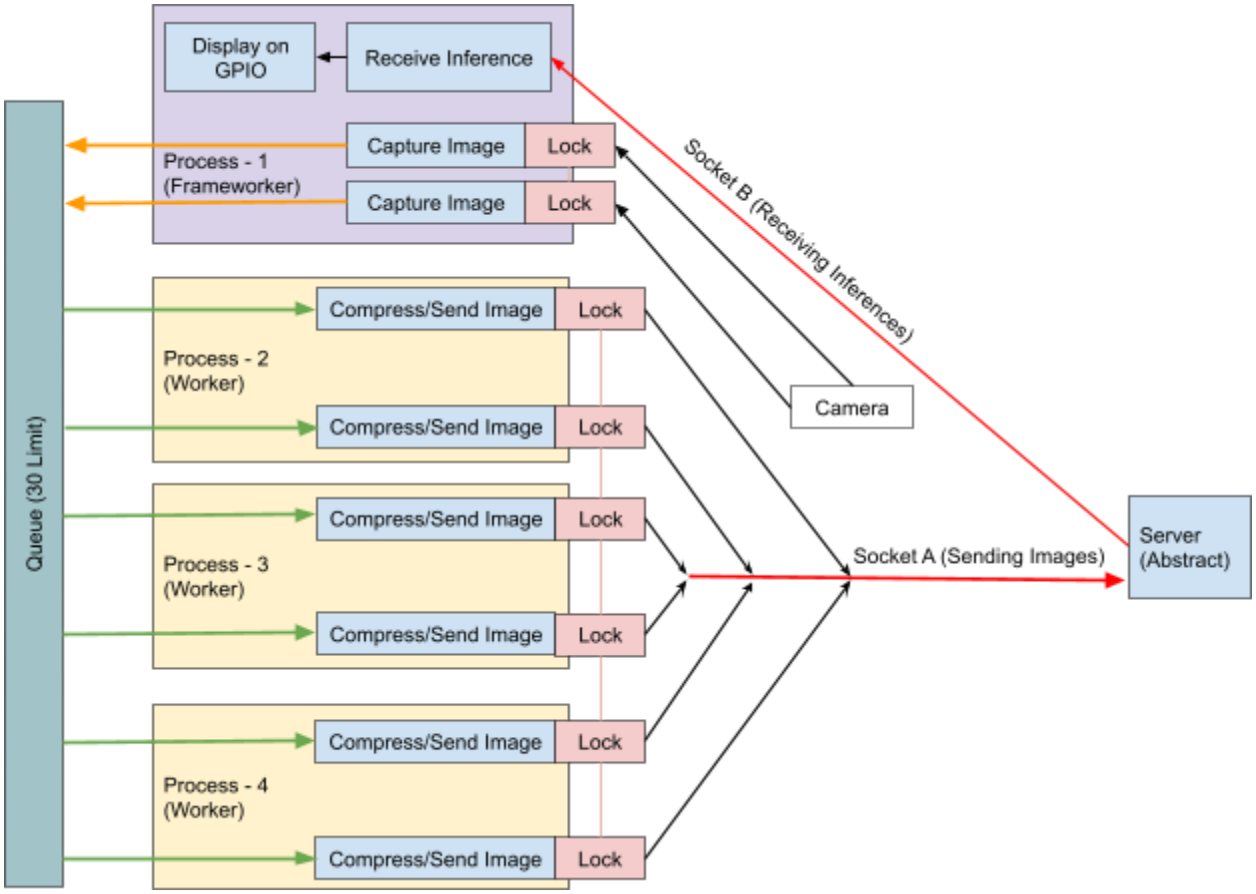


Signed: Shynn Lawrence, Aniket B, Govind G

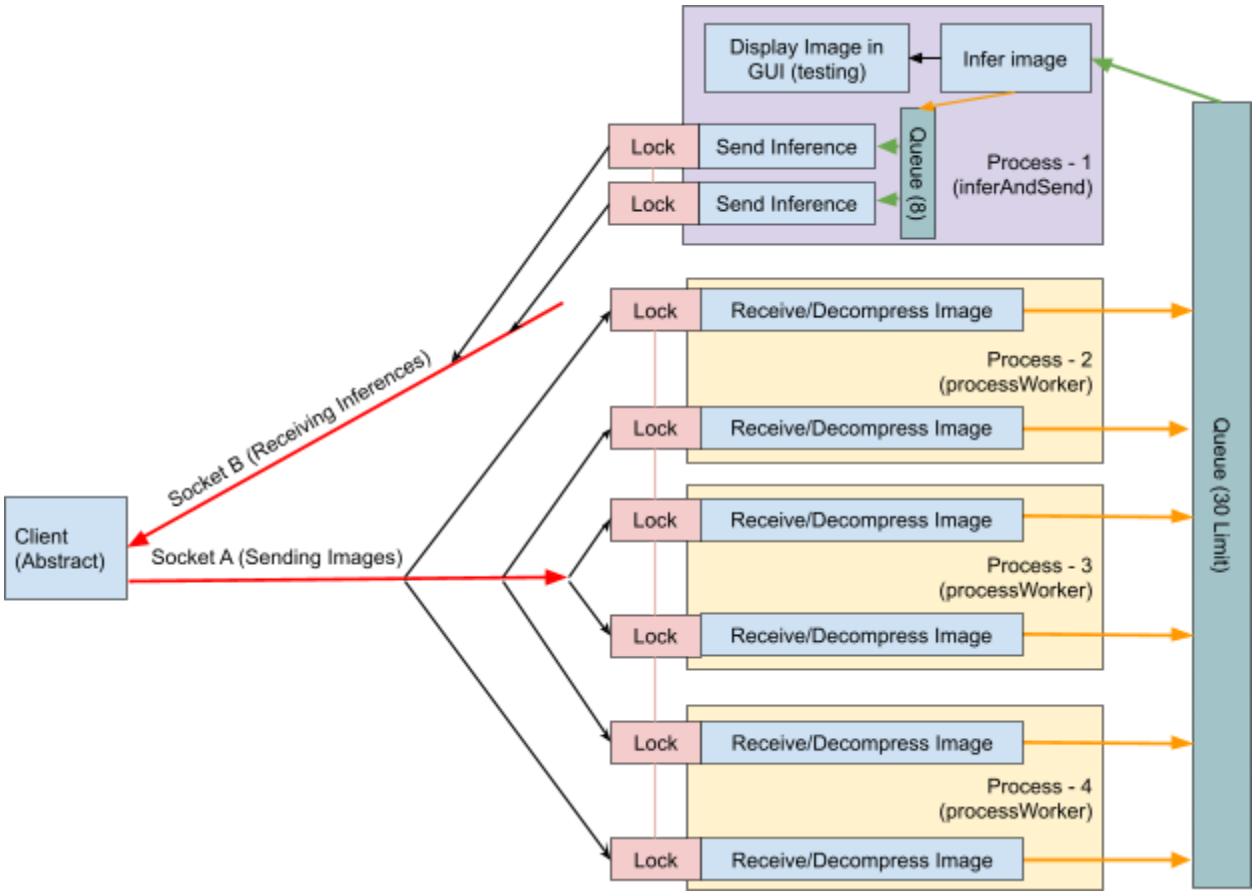


- How and why we measured FPS performance
    - Continuously sent images and stopped when the number of images that passed through the system was 1000 images. The average time that passed between inferences was used to calculate the number of frames per second.
    - Comparing the differing solutions' performance through the use of time (in milliseconds) would be biased, as it would seem as if the synchronous solution was performing better though the asynchronous solution would be spending a long time but doing more operations at once, thus framerate was chosen as a benchmark.

Final Solution's Concurrency Diagrams



The Client-Side concurrency diagram



The Server-Side concurrency Diagram

Conclusion

These past few months have been a wild ride for our team, managing the project on many different platforms, sticking to a schedule, navigating issues, pushing for maximum efficiency, and documenting our work. By the end of this project, we feel a sense of completeness - A problem has been solved and knowledge has been gained.

There is definitely more work to be done - For example, a larger dataset, deployment of the model as a cloud service, and publication of our ongoing android application would be valuable, but the knowledge gained through this project, and the technical skills we learned, will serve us for decades to come.

Sincerely,

Shynn Lawrence, Aniket B, Govind G

Signed: Shynn Lawrence, Aniket B, Govind G

Sources

Garbage Island: The Great Pacific Garbage Patch. (2019, May 23). Retrieved from http://environment-review.yale.edu/garbage-island-great-pacific-garbage-patch

Garbage Problems. (2019, August 31). Retrieved from https://www.planetnatural.com/composting-101/environmental-issues/garbage-gripes/

Garythung. (n.d.). Garythung/trashnet. Retrieved from https://github.com/garythung/trashnet

The Great Pacific Garbage Patch. (2020, February 11). Retrieved from https://theoceancleanup.com/great-pacific-garbage-patch/

How Our Trash Impacts the Environment. (2019, December 16). Retrieved from http://www.earthday.org/how-our-trash-impacts-the-environment/

National Geographic Society. (2012, October 09). Great Pacific Garbage Patch. Retrieved from https://www.nationalgeographic.org/encyclopedia/great-pacific-garbage-patch/

Urban Composting Programs. (2019, August 04). Retrieved from https://www.planetnatural.com/composting-programs/

Signed: Shynn Lawrence, Aniket B, Govind G

Links to tools

Hardware

- Raspberry Pi 3B+
    - Raspberry Pi Cam v1
- Nvidia Jetson Nano
- Samsung Galaxy S10e
- Samsung Galaxy S21

Software

- Python - Software Development
    - Tensorflow - Machine Learning
    - OpenCV - Camera usage in python
    - GPIO Zero - Easy access to GPIO pins
        - RPi.GPIO - Access GPIO pins from python, more low - level (testing)
    - Pillow - Image manipulation
- Android Studio - Application Development
    - Java - Running Applications on Android
    - Kotlin - Translated to JVM instructions to run on Android