ML Assisted Pipeline Crack Detection

### Adam Frost and Brandon Thibodaux

#### 2019.05.07

ML Assisted Pipeline Crack Detection

### Adam Frost and Brandon Thibodaux

# Project Objective

The overall objective of this project is to use machine learning in order to assist in the detection of cracks and anomalies within a pipe based off of sets of data (images) taken by a user. The goal is to have a self contained system in which all computation and controls are handled locally. This will be done by first training the system in order to detect these cracks and anomalies. This trained system will then be able to take in data and specifically separate the data into two categories, damaged and acceptable. From here, the user will then have a database in order to implement machine learning. The final goal will be to take a single picture or set of picture of any given pipe and determine if there is a crack anywhere within the pipe.

# Project Motivation

A heavy motivation for this project was the vastness of the oil and gas industry here in south Louisiana. Pipes are used over and over again for many jobs and many years, and have a long work life. However, time, weather, and sudden traumas can cause pipes to crack or become defected. If these cracks or defections are not found or caught in time, a company can potentially lose large amounts of money as well put many lives in danger due to the pressure and types of hazardous materials that are being transported through these pipes. Another concern is that sometimes these pipes are placed in inaccessible and/or hazardous locations which make for physical checking of certain pipes impossible. We understand that there are better and more advanced ways in order to detect cracks and anomalies in pipes rather than image processing. However, we saw this problem and solution as a way to test and better understand machine learning and its large influence in the business world.

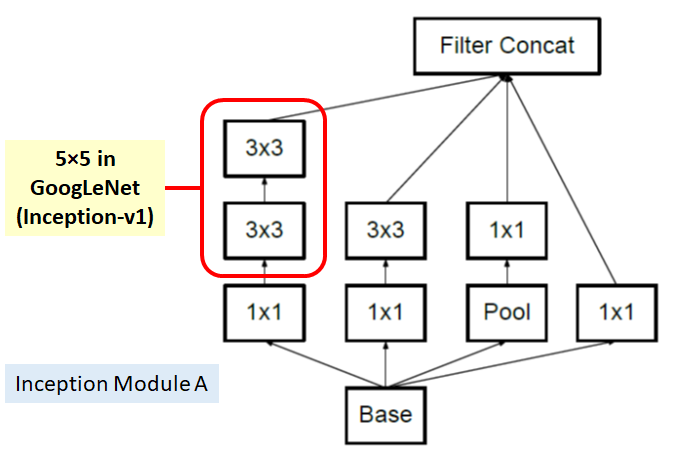
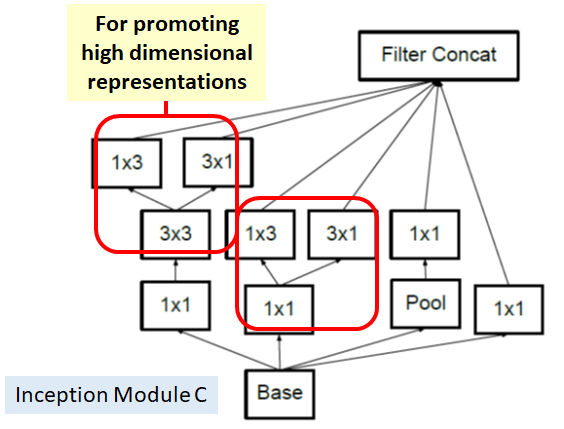
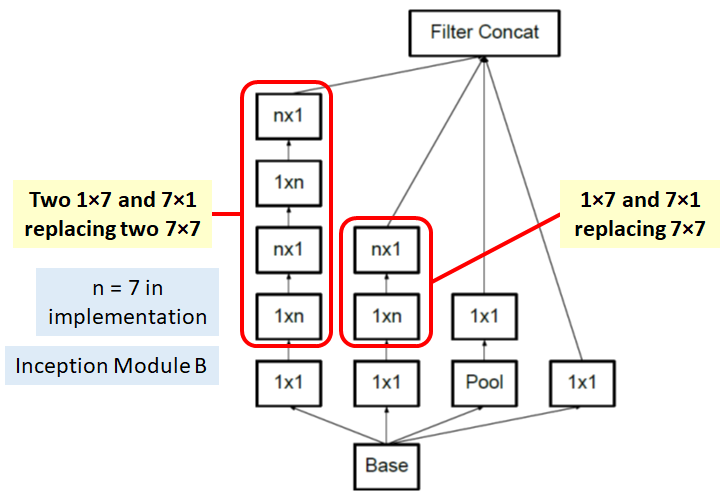
# Brief Literature

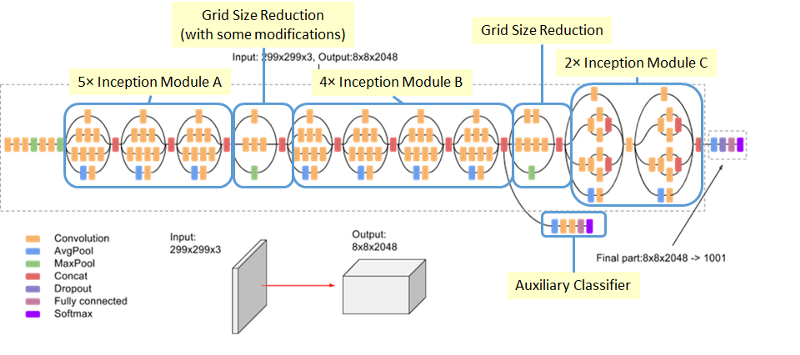
TensorFlow was a crucial and important part of our project and helped us better understand machine learning along the way. In short, TensorFlow is an open source library for numerical computation and large scale machine learning. It bundles together a large amount of machine learning and deep learning models and algorithms and makes them useful by way of a common metaphor. It uses python to provide a convenient front end API for building applications with the framework, while executing those applications in high performance C++.

ImageNet was an important piece to our project. According to imagenet.com,

*“ImageNet is an image dataset organized according to the WordNet hierarchy. Each meaningful concept in WordNet, possibly described by multiple words or word phrases, is called a "synonym set" or "synset". There are more than 100,000 synsets in WordNet, majority of them are nouns (80,000+). In ImageNet, we aim to provide on average 1000 images to illustrate each synset. Images of each concept are quality-controlled and human-annotated. In its completion, we hope ImageNet will offer tens of millions of cleanly sorted images for most of the concepts in the WordNet hierarchy.”*

ImageNet contains 1.2 million training images, 50,000 validation images and 100,000 testing images. ImageNet was used to train the Inception v3 model, which we implemented in our design. Inception v3 is a 42 layer network that can be broken into roughly 3 types of processes, with a focus in reducing the number of parameters needed in the model.



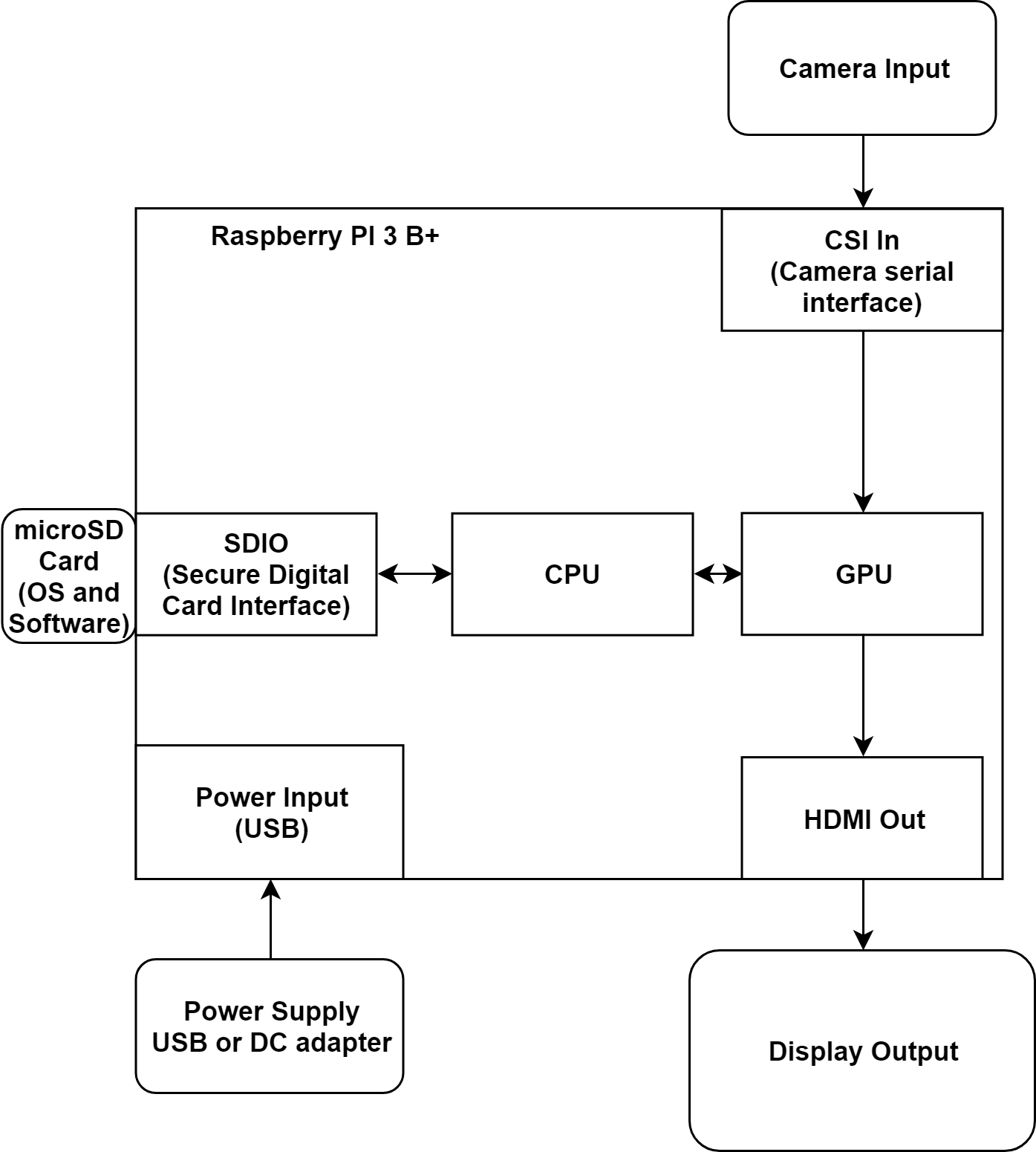
Above Images from: https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c

We used a number of python scripts made available in the “TensorFlow for Poets” example provided by Google CodeLabs. These scripts allowed us to easily retrain an Inception model originally designed to classify flowers, and provided a framework to make edits to create our customized software.

The PYNQ32 was a board that we tested and gained basic knowledge on during the semester and would have been a viable option to use for the project. However, we have more experience with the raspberry pi platform which made the decision to use the raspberry pi over the PYNQ32 more technologically feasible. During testing, it was discovered that the PYNQ processor is only 32-bit, making it incompatible with most of the pre-built software we needed for the project. Since there were no community developed overlays that implemented machine learning for the model PYNQ Z-2, we considered using an image acquisition and processing overlay to speed up the image processing, and have the network run on the processor. This turned out to not be possible with the tools we were able to find. Using the PYNQ32 will be a consideration for future projects.

The original design plans also proposed an acoustic module to be implemented. The idea was that a pipe with defects with have a different acoustic profile than a good-condition pipe. A microphone would be added along with a mechanism to ‘tap’ the pipe and a recording of the sound would be made. The audio could then be processed and fed into a neural network, similar to image classification. However, this turned out to be unfeasible. The setup required a high quality (expensive) microphone, intensive audio processing, and the hardware was already at its limits in processing resources.

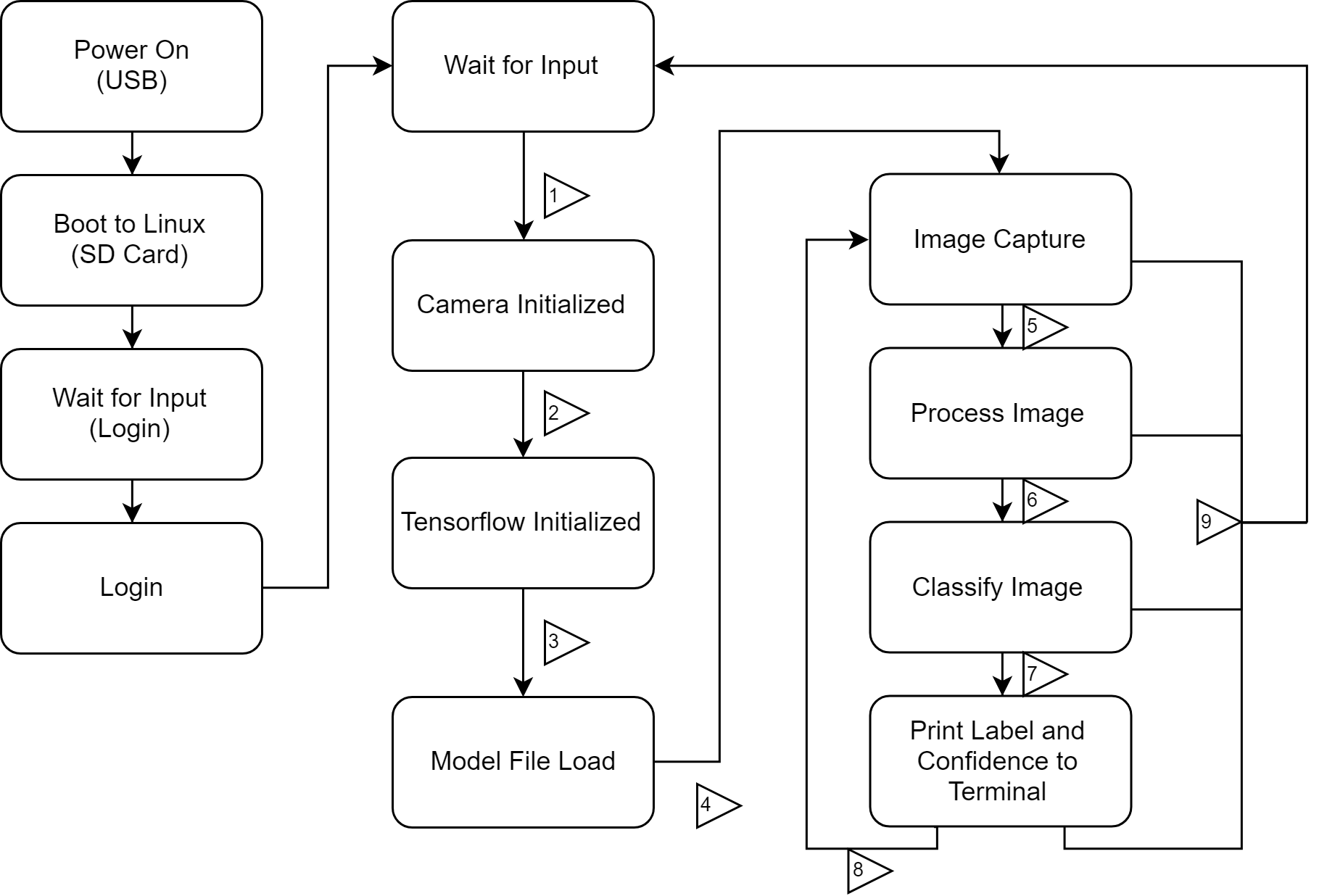
# Project Material

* Raspberry Pi 3 B+
* OmniVision OV5647
* 16GB+ Class 10+ microSD card
* Raspbian Linux Distro (Pi)
* TensorFlow 1.13.1-GPU (Desktop Training) (Windows 10)
  + CUDA 10.0 (Windows 10)
  + cudNN 9.0 (Windows 10)
* TensorFlow 1.13.1-ARM (Pi inference)
* Anaconda3 Python Environment Manager (Windows 10)
* IfranView (Batch image file editing) (Windows 10)
* PVC Pipe

Project Block Diagram

# 

# Project State Diagram



1. Run label\_image.py script
2. Camera finishes warm-up
3. Dependencies loaded, resources finish allocating
4. output\_graph.pb finishes loading into TensorFlow
5. Raw RGB data loaded in buffer
6. Processing completes (Image resize, formated to JPG)
7. Image finishes passing through NN
8. No new input
9. User or System Interrupt

# 

# Machine Learning Model

There were two phases to this project:

1. Testing/Research
2. Implementation

During the first phase, we determined how we were going to go about the machine learning aspect of the project and tested this process out. We decided to use TensorFlow. We first made sure to pick a program in order to edit the Python scripts of TensorFlow. We decided to go with VS Code. We then decided to use Anaconda3 in order to create a sandboxed Python environment to work in. Once inside the created environment, we then installed TensorFlow using pip. We then went online and used a prepackaged example (https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/#0). We used the example’s existing data set to build our Inception model file. This data set was composed of images of five different flowers: daisys, dandelions, roses, sunflowers and tulips. We tested the created model using images of flowers found online. Once we knew the model was being successfully trained, we moved on to create our own classification.

During the second phase, we took the knowledge we obtained from the first phase and the assurance that this process worked and then implemented our project. The first part and most important part was building a dataset to work with. We first took a piece of clean PVC pipe and used our phone in order to take a timelapsed image sequences of the of the pipe. We kept the phone stationary and rotated the pipe slowly and at different angles and distances in order to prevent blurry pictures. We then repeated this process but this time with a “cracked” piece of pipe. To mimic a cracked pipe we used a sharpie and drew various lines on the pipe. The end result of this step was to obtain two separate files/datasets: “Damaged” and “Acceptable”. Next, we took these images and converted them all to 640x480. We used the program IrfanView for this process. We then took these two folders and placed them inside one folder. After this, we then placed this folder into our previous example and ran through the same process. However, instead of having flowers, we had our pictures of the damaged and accepted pipe images. Once the the system learned the difference between a cracked and normal pipe, we were able to take a single, new picture of a pipe and have the system correctly identify if the pipe is damaged or not.

Once the training and inference was complete, we had to deploy the inference onto our embedded system. This process took a lot of research and troubleshooting, and there were issues with getting our model, created in an x86 environment, to run in an ARM environment. The problem was not the model file itself, but version mismatches with TensorFlow, even though they were both labeled as “1.13.1”. In the end we were successfully able to run inference on our Pi. We then modified the inference software to keep TensorFlow and the model loaded and looped camera controls to take a new picture after each inference result was printed to the terminal.

# 

# Future Work

* Larger dataset
  + Increase background variety until NN ignores it
* Hardware upgrade:
  + - Implement object detection to check for a pipe, remove the background, then run the pipe crack inference
    - Faster sample and inference rates
    - UI with live preview, bandboxing, classification and confidence % overlays
    - Better camera, increase image resolution, finer cracks detectable
    - Lighting to improve image capture detail
* Automation
  + Mobility - self-propelled and guided
  + Map pipeline, crack/damage heatmaps
* Other tests
  + Detect rust/stains and other signs of cracking
  + A device using similar hardware and software, but in a package that allows interior pipe work. Would require adaptable locomotion, camera gimbal/servos, weatherproofing, lighting, remote control, and batteries.

# Roles:

Adam - inference, hardware, research

Brandon - Dataset building, training

# Challenges

* Software compatibility/interactions

We estimate 85% of the project’s time was spent troubleshooting and learning how to use the software rather than implementing it into our design. Many different pieces of software needed to come together and interact for this project to work. Most ML design is done in Linux environments with x86 processors, but we were using Windows x86 and Linux ARM environments. There were compatibility issues early on that required lots of guess and checking to find versions of packages that worked well together, and some of these packages had to be built from scratch before they could be tested. On the Raspberry Pi, this can take a very long time, so it significantly slowed down testing.. However, once we started to get things working, the speed of the final design’s progress picked up significantly. Future work in embedded ML systems should be much faster.

* Limited Python knowledge

Another challenge that was presented over the course of the project was the teams inexperience with the program language Python. This lack of experience caused a roadblock in the efficient progress of the overall goal because the environment in which the project worked out of was Python based. All of the pre written scripts and examples were written in Python, which helped us as well has slowed us down. The understanding of the scripts took a little time to understand and would have been little smoother if we had previous experience with Python. It also would have allowed more customization of the software being run as well as allowed a more feature-rich final design.

* Limited TensorFlow knowledge

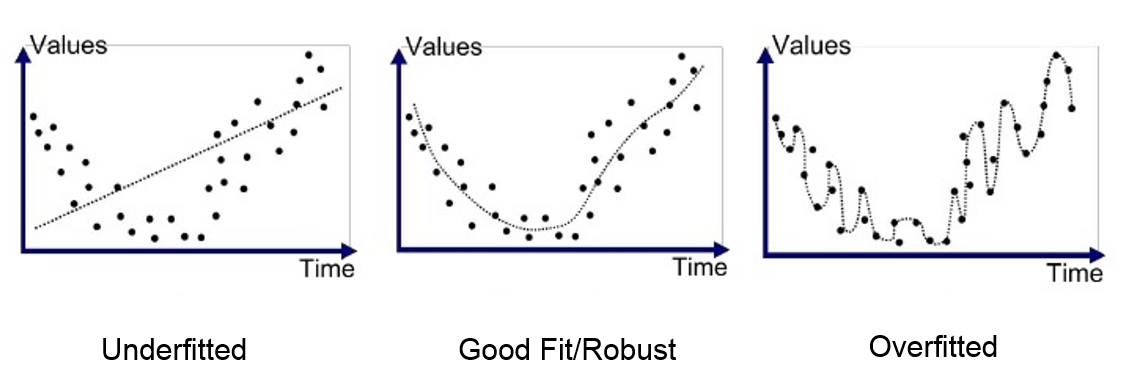
As stated previously about the lack of experience with Python as being a hurdle in the project, the same can be said for TensorFlow. Although this was a staple in the machine learning aspect of our project, we had little previous knowledge of TensorFlow code. Anytime this lack of knowledge is associated with an aspect of a project, time is dedicated into becoming familiar and comfortable with said topic rather than designing and implementing features.

* Can ML overcome dirt/staining/paint with a large enough dataset

A physical challenge that will be presented in future applications outside of a controlled piece of pipe is the ability to detect and overcome the physical aspects associated with the pipe outside of the cracks. With a large enough data set, could we possibly train the system in order to ignore things such as dirt, staining and even markings on a piece of pipe? This would take a substantial amount of data in order to detect these other anomalies and would be a great starting point on the future of this project. Similarly, a larger data set that ignored unimportant surface features like dirt could also allow for a wider variety of pipes to be tested. Currently, the model works well with PVC, but is untested on other types of pipe.

* Overfitting

With such a small dataset (~3000 images) it was very easy to overtrain the model to the point of uselessness. The model would become an expert at identifying the images loaded, but would fail when presented with new images. The final model performed around 3500 steps.

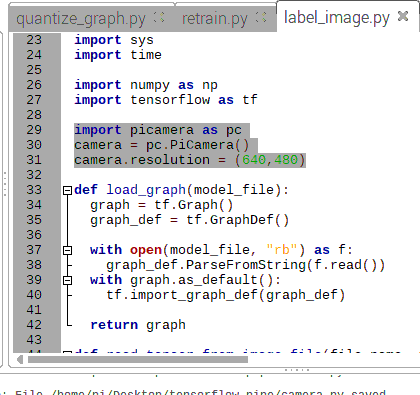


# Appendix A: Code Used

Our code is based on the example from Google CodeLab’s “TensorFlow for Poets” (links below) . We used this as a guide and framework to build and retrain our Inception network. The original classification script loaded TensorFlow, loaded the model, loaded the image, processed the image, classified the image, print the results to the terminal, then unloaded everything. We modified this script to keep TensorFlow and the model file loaded indefinitely (until the program is interrupted by the User). In addition, we added code to load, initialize and set the parameters for the Pi’s camera. After all the components were initialized, we had a loop to take a picture, save it to a location the classifier knew to look, classify the image, output the results, then take another picture, classify, and so on.

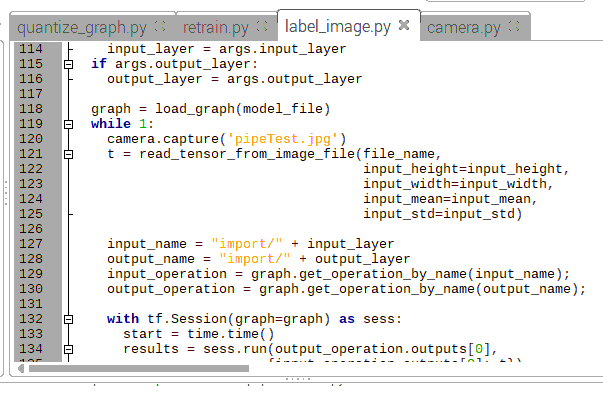
https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/#0 https://github.com/googlecodelabs/tensorflow-for-poets-2

The following is the code added to the “lable\_image.py” file:

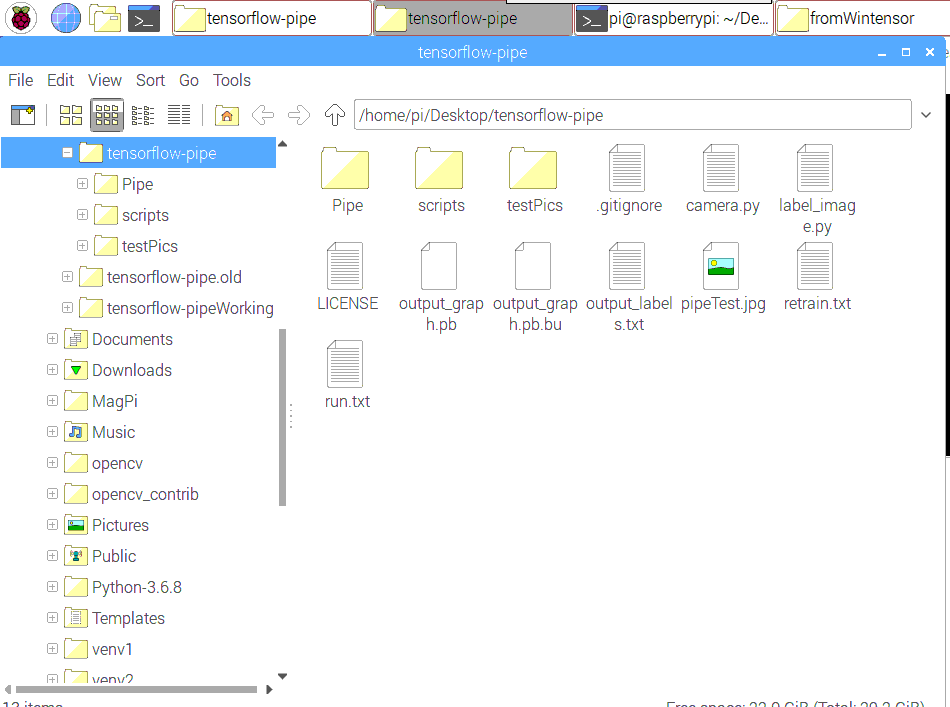
Camera:

import picamera as pc  
camera = pc.PiCamera()  
camera.resolution = (640,480)

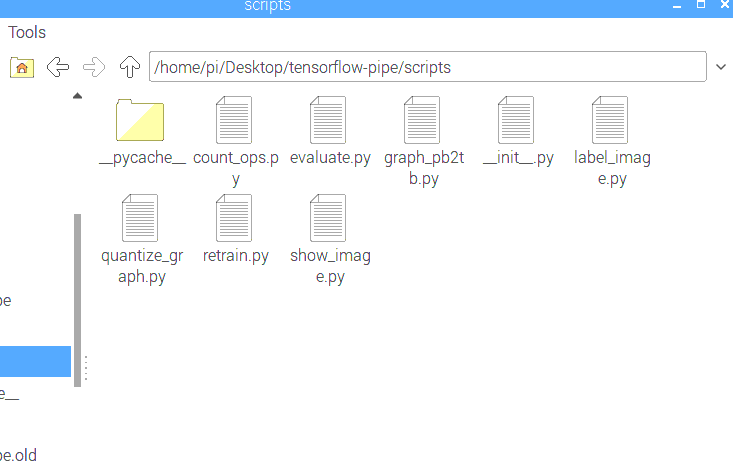
Loop to keep model loaded:



# Appendix B: Project Images



Project Folder   
Structure:



Script Folder  
Structure:

Raspberry Pi running camera and image classification:

