SnapCrack

Akira DeMoss - Lead Deep Learning Engineer, Meeting Facilitator

Maggie Dalton - Lead Software Engineer, Meeting Scribe

Modeste Kenne - IoT Engineer, Test Engineer

Nik Thota - Software Engineer, Report Manager

sdmay20-18

Client: Bo Yang

Adviser: Halil Ceylan

Team email: sdmay20-18@iastate.edu

Team website: http://sdmay20-18.sd.ece.iastate.edu/

Revised: 12/8/2019 (Final)

Executive Summary

Development Standards & Practices Used

- Documentation According to Class Quality
- Life Cycle Processes - Risk Management
- Software Reliability Standards
- Guide for Taxonomy for Intelligent Process Automation Product Features and Functionality

Summary of Requirements

- Create algorithm that identifies cracking, longitudinal cracking, and potholes
- Evaluate severity level of cracking in accordance with LTPP distress manual
- Data collection done by smartphone
- Severity levels defined for 3 different types of pavements:
    - Asphalt concrete surface
    - Jointed portland cement surface
    - Continuously reinforced concrete surfaces.

Applicable Courses from Iowa State University Curriculum

- CprE 388 - Embedded Systems II: Mobile Platforms
- Com S 309 - Software Development Practices
- Com S 311 - Algorithm Analysis & Design
- Com S 319 - Construction of User Interfaces
- Com S 363 - Database Management

New Skills Acquired Not Taught in Courses

- Machine Learning
    - Setting up/using TensorFlow
    - Labeling data sets
    - Training algorithms
- Servers
    - Creation
    - Management
    - Interactions with other platforms

Table of Contents

List of Tables and Figures

1. Introduction

1.1 Acknowledgement

We would like to acknowledge our client, Bo Yang. Bo has been working with us from many angles to progress on this project, and will likely be a large contributor to our success. The basis for this project is an unfamiliar topic, but Bo has been guiding us in how to properly classify cracks based on the LTPP manual. In addition to the help with LTPP, he has forwarded research to us that is relevant to the project. He has also expressed that he will help us in the event that we need financial assistance or additional hardware.

1.2 Problem and Project Statement

1.2.1 Problem Statement

The classification of cracks and potholes found in roads and their severity currently requires manual measurement of the affected area. This measurement requires researchers to leave their vehicles and enter the roadway, which is often dangerous. Measurements are also often time consuming to obtain.

1.2.2 Proposed Solution

The purpose of this project is to develop an Android application that removes the need for a researcher to leave their vehicle when classifying cracks/potholes based on the type. The project is driven by the need described above. This measurement is a vital part of determining where to focus research and gather additional data. The current process is both slow and dangerous; researchers are often standing alongside lanes of high speed traffic.

Our planned solution is a mobile device running a mobile application running a trained machine learning algorithm which will be mounted on the dashboard or windshield of a vehicle. The app will inference whether the stretch of road ahead of the device contains a pothole or crack. From there, the mobile device will snap an image and send it to a remote server along with the

location of the image. From there, the image will be accessible along with location on a web-based portal.

1.3 Operational Environment

The general operational environment for this project will be on a mobile device mounted on either the dashboard or windshield of a car using a 3rd party mounting device. Although the device will not withstand any extreme weather conditions, the user will not be required to interact with it while driving. It must be capable of running automatically after minimal user set-up prior to driving, and operate in a non-distracting manner to the driver until manually terminated.

1.4 Functional Requirements

1.4.1 Android Application

1.4.1.1 The application will use a trained model to detect longitudinal cracking, transverse cracking, and potholes found in roads.

1.4.1.2 If a crack/pothole is detected the app will save an image of the area where the detection occurred to the mobile device.

1.4.1.3 If a crack/pothole is detected, the app will save the location data, date, and time of when/where the detection occurred.

1.4.1.4 If the user is connected to the internet, the application will submit collected images and location data to the server to be stored in the database.

1.4.1.5 If the user is not connected to the internet, the application will maintain a list of stored images and relevant data that have not been submitted to the server.

1.4.2 API

1.4.2.1 When the server is shut down the API shall return appropriate error codes detailing the loss of the server.

1.4.3 User Interface

1.4.3.1 The user can press a button to make the app start scanning the road for cracks/potholes. After the button is pressed the app requires no further interaction from the user to detect cracks/potholes.

1.4.3.2 The user can press a button to stop the scanning process.

1.4.3.3 The app will display bounding boxes around detected cracks/potholes to provide a visual feedback of detections.

1.4.3.4 Given the API is down, the user interface shall remain functional.

1.5 Non-functional Requirements

1.5.1 Android Application

1.5.1.1 The detection shall be able to detect cracks/potholes on multiple road types (composite/asphalt/etc.)

1.5.1.2 The app shall be capable of detecting cracks/potholes at various speeds

1.5.1.3 Detections made by the app shall comply with the LTPP distress manual.

1.5.2 API

1.5.2.1 The API shall have documentation for each endpoint.

1.5.3 User Interface

1.5.3.1 If the classification server is not available, the user interface shall display that there was a loss of communication to the user.

1.5.3.2 If an error occurred while sending or receiving images/videos from the server, the user interface shall display a user-friendly error message.

1.5.3.3 The user shall be capable of viewing past detections along with their data.

1.6 Intended Users and Uses

The intended users for the deliverables from this project are researchers. Researchers in this area require measurements of cracks and potholes in the road, but gathering those measurements are both time consuming and dangerous. By using this project, researchers will be able to gather images and classify cracks/potholes much quicker than they could manually. Researchers will also be able to complete this task from their vehicle, which will be safer than roadside. They prioritize accuracy and availability of data, so we will focus on those two ideas when implementing our project.

1.7 Assumptions and Limitations

1.7.1 Assumptions

1.7.1.1 The mobile device will be mounted using a third party mounting device on either the dashboard or windshield

1.7.1.2 The application will be started prior to the user driving, requiring no interference until the user has finished evaluating their target area

1.7.1.3 The product will not be used outside of the United States

1.7.1.4 There will be 10 or fewer concurrent users sending/receiving data from the server at a given time

1.7.1.4.1 Primary focus as defined by the client is on creating a functioning and accurate algorithm and less on supporting multiple users

1.7.1.5 Data obtained from the application and any processing of the data will be stored in a centralized database

1.7.1.5.1 We don't currently foresee different organizations using this application simultaneously

1.7.2 Limitations

1.7.2.1 The application should be run on commercially available mobile devices without the use of additional hardware (outside of the mounting device)

1.7.2.2 The application should be simple to use, requiring no prior knowledge of machine learning, LTPP crack classification, or any other prior knowledge to operate

1.7.2.3 The classifications will be categorized using criteria from the LTPP distress manual

1.8 Expected End Product and Deliverables

1.8.1 Android Application

The Android application shall be free and available through a downloadable APK. It shall require no additional downloads or modifications to the Android device to be operational. It will operate exclusively with the delivered server and database using an API designed for this project. This application will be capable of using a trained model to detect cracks/potholes

found in roads and displaying information pertaining to the detection for the user.

1.8.2 API

There will be an API that can be used to communicate with the server to create, read, update, delete, and query data stored on the database.

1.8.3 Database

The database will be made available in a condition that requires no additional actions from the client to operate. Instructions for how to remove data or manually add data to the database will be included in project documentation.

1.8.4 Server

The server will be available in a condition that requires little to no additional actions from the client to operate. Actions such as restarting or terminating the server will be simple to execute. Instructions for how to restart the server in the event of an issue and a list of error code definitions will be included in project documentation.

1.8.5 Trained Detection Model

A trained model will be included in the Android application for the detection of cracks/potholes. Detectable crack types will include transverse cracking, longitudinal cracking, and potholes as described in the LTPP manual.

1.8.6 Documentation

Documentation for the process of using each component and making general expansionary modifications will be made available. This will include, but is not limited to, restarting each component, wiping each component of data and restarting, performing general maintenance, and error handling. Documentation for how to expand upon the types of classifiable cracks may also be included.

2. Specifications and Analysis

2.1 Proposed Design

We've identified three loose approaches to this project; an all-in-one mobile application, a mobile application that sends data to a server for complete detection/classification with

a web portal, and detecting using the Android application while making data available in a web portal. Each approach has pros and cons. The all-in-one application would be the simplest and least data-intensive option for the end user, but may function too slowly to detect cracks/potholes at higher speeds. On the other hand, doing all of the detection server side would be extremely accurate but require sending large amounts of video or images to the server, and thus be extremely data intensive. Lastly, a blended approach would allow us to detect cracks/potholes at a relatively quick pace while creating a more user-friendly portal for viewing collected data. While we recognize the value in the other two approaches, we settled on a blended application. Our current proposed design is as follows:

2.1.1 Mobile Application

2.1.1.1 Runs a trained machine learning model for detecting potholes/cracks using TensorFlow Lite

2.1.1.2 Captures images when a potential crack or pothole is detected

2.1.1.3 Sends the image, date, time, and location data to the server either as detections are made or in a batch at user request

2.1.2 Server

2.1.2.1 Receives the image and relevant data from the mobile application

2.1.2.2 Stores the original image and relevant data in a database

2.1.2.3 Makes the image and relevant data accessible to the web portal

2.1.3 Database

2.1.3.1 Stores the captured image, location, date, time, and a unique identifier for each detected crack/pothole

2.1.4 Web-based User Interface

2.1.4.1 Displays collected images, location, date, and time for each detection

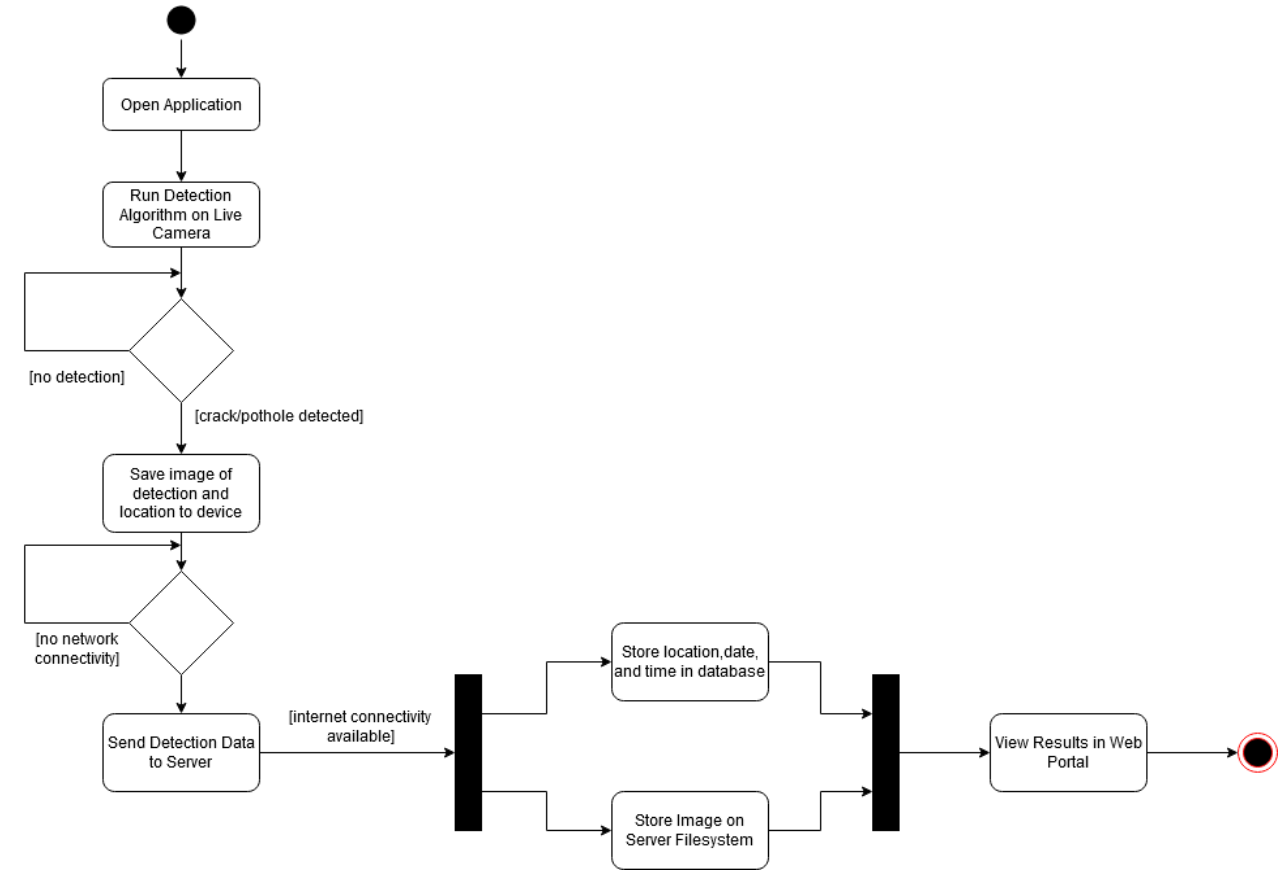


Figure 1: Activity Diagram

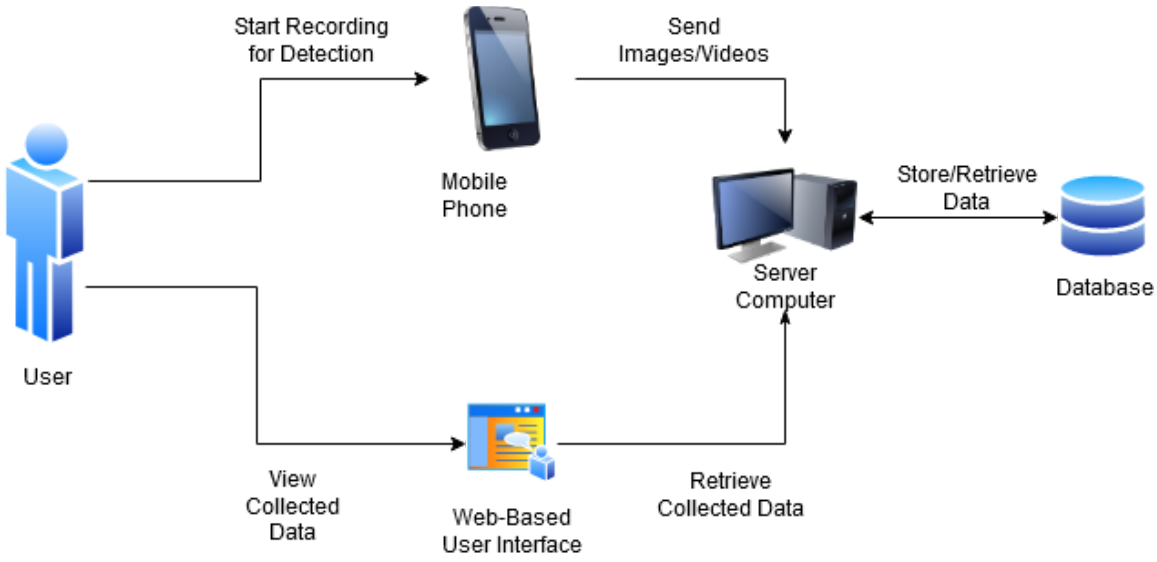


Figure 2: Deployment Diagram

2.2 Design Analysis

With this design approach we were able to combine the strengths of the other two options. We believe that doing detection in-app will cut down on the amount of required data. By detecting cracks/potholes as we go we can keep only relevant data, and therefore less storage is required both on the device and in the database.

It is also worth noting, however, that this design came with some concerns. While we are satisfied with how a mobile device makes inferences, making detections server-side on videos would allow for more accuracy at high speeds. With a Pixel 4 XL the device was able to make detections at about 7-10ms per inference. Limitations in mobile hardware with both the camera quality, camera speed, and processing power may inhibit performance on some devices.

Another large design choice we have made was to develop the application for Android devices. Our group has significantly more experience with creating Android applications and each member owns an Android device. Helpful frameworks, like TensorFlow Lite, have been created by Google and also have greater support and documentation for Android.

Initially, we were apprehensive of the blended approach being attainable, but as we continued to research and implement prototypes it increasingly proved to be a valid option. We found an offshoot of TensorFlow named TensorFlow Lite that is optimized for mobile. After working through some examples of TensorFlowLite YOLO detection, we felt with some adjustments the application would be able to inference quickly.

2.3 Development Process

We followed an Agile approach to this project. Our sprints were roughly two weeks in length and included biweekly demos with our client. Tasks were organized on Trello with task designation occuring during weekly team meetings. We used GitLab for organization and Google Drive for organization and creation of documentation.

2.4 Engineering Standards & Design Practices

We used a handful of design practices to ensure maintainability and help foster a smoother development process.

2.4.1 Standardized Tooling and Technologies

The tools and technologies used in this project were evaluated and chosen based on both their effectiveness, and the ease of use and implementation by team members. By choosing a standardized set of tools and technologies that all members were comfortable with, we were able to aid each other much more quickly when issues arose.

2.4.2 Process Standards

Early on in the planning stage we agreed upon a process which we could all follow when developing this project. Tasks were divided into cards on Trello, which were assigned in meetings, completed by members, discussed in following meetings, and reviewed before being merged with completed tasks.

2.4.3 Standardized Coding Style

To aid readability and maintainability of this project, our group created a document that outlined a standardized coding style. This document includes information such as naming conventions, commenting conventions, and indentation.

2.4.4 Code Quality Control

Before code is merged to the overall project, it is reviewed by at least one other group member for adherence to the standardized coding style.

2.5 Design Plan

The mobile application will be designed for Android targeting API level 27 or later to allow the use of the existing Neural Networks API designed by Google. It will utilize TensorFlow Lite to make inferences using a trained model. The model will be trained to detect transverse cracks, longitudinal cracks, and potholes. Detections will be made on the device using the model and back-facing camera, and collected images will be stored on the device until the device is connected to the internet. If internet connectivity is available, the device will send images to the server that have not been sent already using the Volley library.

The server was with Node.js and utilized the API to store data in a MySQL database. If the API or server were to shut down for any reason, the data will remain uncorrupted and the Android and web interfaces will display appropriate error messages. Following the reception of images, the server will store the incoming data in the database.

The web interface will allow users to view and retrieve existing data from the database, including images, location, date, and time in an easy to read format. From the web portal, if the user disagrees with the classification of a crack/pothole, they can either manually delete the detection or manually change the type of the crack/pothole.

3. Statement of Work

3.1 Previous Work And Literature

Previous work demonstrates advances in object detection algorithms have lead to research leveraging deep learning for road damage. From the literature there were three popular methods for realizing this task; the first leverages a simple convolutional neural network (CNN) architecture which is used for inference to generate simple bounding boxes during run time. The second leverages more complex CNN architectures which semantically segment the objects from the background. Finally, the last provides an alternative approach to crack detection using a conditional Wasserstein Generative Adversarial Network (WcGAN). For our project we are specifically trying to detect potholes, longitudinal cracks, and transverse cracks in concrete. Furthermore, our task involves detecting the severity level of each crack, as well as using a smartphone for data collection. The combination of these three requirements has yet to be realized.

First, from the article titled "Road Damage Detection Using Deep Neural Networks with Images Captured Through a Smartphone"[1], the authors created their own custom dataset by mounting a smartphone to the dashboard of a car that takes photos every second. From this dataset they collected over 9,000 images in which they used alongside the SSD Mobilenet architecture to train weights and biases for object detection. SSD is an object detection framework which uses a single feed-forward CNN to directly predict classes and anchor offsets without requiring a second stage per-proposal classification. The authors implement SSD using both Mobilenet and Inception V2 and conclude that implementing SSD using mobilenet on a smartphone is the faster option of the two.

Second, the article titled "A Deep Learning Approach for Road Damage Detection from Smartphone Images"[2] uses the same dataset to evaluate road damage, however instead of using the SSD architecture they use YOLO. This article YOLO, however their results are not tested on a mobile device. Our research has determined that in order to get this working in a smartphone, a lighter weight version of YOLO is required to be used to derive the weights that will map the input data to our desired bounding box detection outputs.

Next, in the article titled "Road Damage Detection and Classification with Faster R-CNN"[3], they use pixel-level segmentation to annotate and identify cracks in the pavement. While this approach is more accurate, it is more computationally expensive. Finally authors in the article titled "Road Damage Detection and Classification with Faster R-CNN" leveraged an approach using a conditional Wasserstein generative adversarial network (WcGAN) is used. Using this technique the authors were able to generate synthetic training data which would allow a deep learning practitioner to train an

object detection model with the generated images to improve the detectors accuracy, resulting in a highly accurate object detector.

For our project we are specifically planning to detect potholes, longitudinal cracks, and transverse cracks in concrete. Furthermore, our task involves detecting the severity level of each crack, as well as using a smartphone for data collection. The combination of these three requirements has yet to be realized.

3.2 Technology Considerations

Implementing a convolutional neural network (CNN) to train your own custom object detector is a highly complex task we will achieve through a combination of the right operating system, hardware, and software. We have strategically chosen a technology stack which will fulfill the requirements of the project and be easy to pick up for people new to machine learning with the help of our extensive documentation.

Linux Ubuntu 18.04 is used to build our development environment where we will manage all of our third party libraries used for training the weights and biases of our object detectors. Linux Ubuntu 18.04 has an active community of support for deep learning practitioners and is great for establishing a development environment for deep learning.

A GeForce 930MX Graphics Processing Unit (2GB) will be used for intensive processing during training. For our initial prototype we fed over 9,000 images into a custom developed yolov2-tiny object detection architecture that we used to train 3 classes that detect transverse cracks, longitudinal cracks, and potholes respectively. With this many images and multiple classes, the task of training weights for inference requires the matrix multiplication of millions of parameters stored as n-dimensional matrices. GPU's allow thousands of threads to be run concurrently in contrast to one or two threads per CPU core which exponentially reduces training time.

Our prototype implementation of our object detector was specifically trained and ran using CUDA, OpenCV, and Tensorflow. CUDA enables GPU acceleration that facilitates the processing-intensive matrix multiplication operations involved in training a custom object detector. OpenCV performs inference and generates the bounding boxes which display the predicted object and confidence level of this prediction.

3.3 Task Decomposition

3.3.1. Gather Requirements

3.3.1.1 Gather functional requirements

3.3.1.2 Gather non-functional requirements

3.3.2. Gather Domain Knowledge

3.3.2.1 Gather information on problem context

3.3.2.2 Gather information on data collection mechanism

3.3.3. Implement a Custom Object Detector

3.3.3.1 Choose multiple CNN architectures to train and to test

3.3.3.2 Setup the development environment to train and run inference

3.3.3.3 Extensively document installation steps

3.3.3.4 Using multiple YOLO architectures to train road damage detectors

3.3.3.5 Run tests on custom object detectors

3.3.4. Create a Custom Dataset for Object Detection

3.3.4.1 Identify pre-existing datasets for road damage detection

3.3.4.2 Integrate pre-existing datasets

3.3.5. Scale a Custom Object Detector to a Mobile Device

3.3.5.1 Collect data for custom dataset for road crack detection

3.3.5.2 Create a custom dataset

3.3.5.3 Install Tensorflow

3.3.5.4 Extensively document tensorflow installation instructions

3.3.6. Test the Object Detection System's Ability to Generalize

3.3.6.1 Collect data for custom dataset for road crack detection

3.3.6.2 Create a custom dataset

3.3.6.3 Install Tensorflow

3.3.6.4 Extensively document tensorflow installation instructions

3.3.6. Implement UX / UI Design

3.3.6.1 Design user friendly AI for Android application.

3.3.6.2 Parse GPS coordinates from image metadata

3.3.6.3 Create web based UI for severity level recognition

3.3.7. System Level Testing

3.3.7.1 Test and measure the speed and accuracy of object detection

3.3.7.2 Test the integration of Android, the server, and web-based UI

3.4 Possible Risks And Risk Management

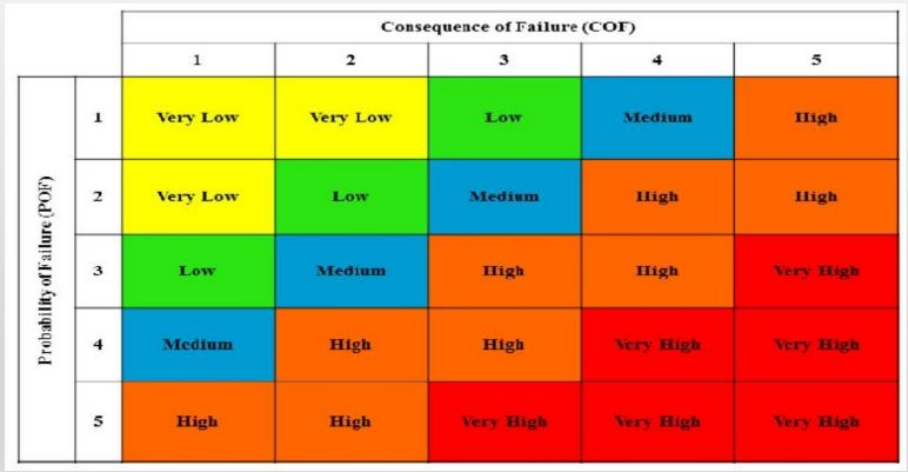
Risk Occurrence Scale

1. Improbable: unlikely to happen - frequency: **0**
2. Remote: unlikely but possible - frequency: **0.01**
3. Occasional: could happen sometimes - frequency: **0.05 - 0.1**
4. Probable: is expected to happen - frequency: **0.1-0.3**
5. Frequent: will occur several times in the year: **1**

Risk Impact Scale

1. Negligible:
2. Minor: late **1 week** with **no penalty**
3. Serious: late less than **2 weeks** with **10% penalty**
4. Critical: late less than **3 weeks** with **20% penalty**
5. Catastrophic: late more than **4 weeks** with **30% penalty**

Risk Exposure Matrix



Probability of Failure (POF)	Consequence of Failure (COF) 1	2	3	4	5
1	Very Low	Very Low	Low	Medium	High
2	Very Low	Low	Medium	High	High
3	Low	Medium	High	High	Very High
4	Medium	High	High	Very High	Very High
5	High	High	Very High	Very High	Very High

Table 1. Risk Exposure matrix

Risk Evaluation

Risk	Occurrence	Impact	Exposure
Hard drive fails - lose training software	0.01	2	Very Low

Table 2. Risk Evaluation

Risk Breakdown

Title: Hard drive fails - lose training software
Response: Risk Reduction
Justification: Overall our project plan is very well thought out and we are confident that we'll be able to achieve our goals, the only risk that our team could think of would be if our hardware where all of the training software was installed failed at some point that this would set us back about a week or so in installation times. For this reason we documented all of the steps for installing Cuda, Tensorflow, and OpenCV.
Mitigation Action: Documentation on installation steps of all open source software compiled from sources has been created.

3.5 Project Proposed Milestones and Evaluation Criteria

Milestone 1: Preliminary Phase is Complete

- Functional requirements gathered.
- Nonfunctional requirements gathered.
- Cuda 9.0 + cudnn v7.3.0 and Darknet installed
- OpenCV Version 4.0.0 installed.
- Tensorflow r1.12 installed.
- Installation steps documented for all 3rd party software.

Milestone 2: System Design is Complete

- All parts designed for networking.
- All parts for user interface are designed.
- Create database diagram.
- Create high level systems level diagram.

Milestone 3: System is Constructed

- Object detection integrated into Android smartphone app that detects initial cracks and potholes.
- Web UI allows clients to view the results of server-side object detection.
- All parts of the system are integrated.

Milestone 4: System Testing is Complete

- Data collection has been tested using smartphone mounted in car.
- Testing has been completed to verify an image can be sent to the server.
- Testing has been completed to verify the processed image can be viewed on the Web UI.

3.6 Project Tracking Procedures

Our group is using a Trello board to track work that is in progress, on the backlog, and completed. In addition to the trello board we are using GitLab issues. We provide our team with a standard template to fill out when reporting issues to the repository that enables easy communication and collaboration in resolving these issues.

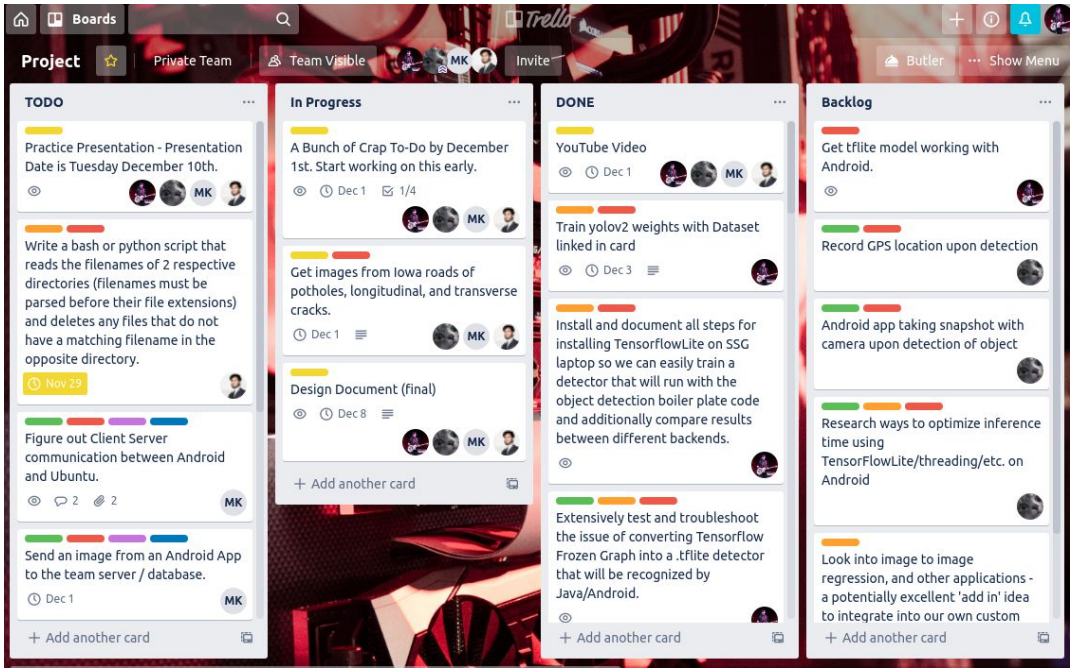


Figure 3. Trello Board

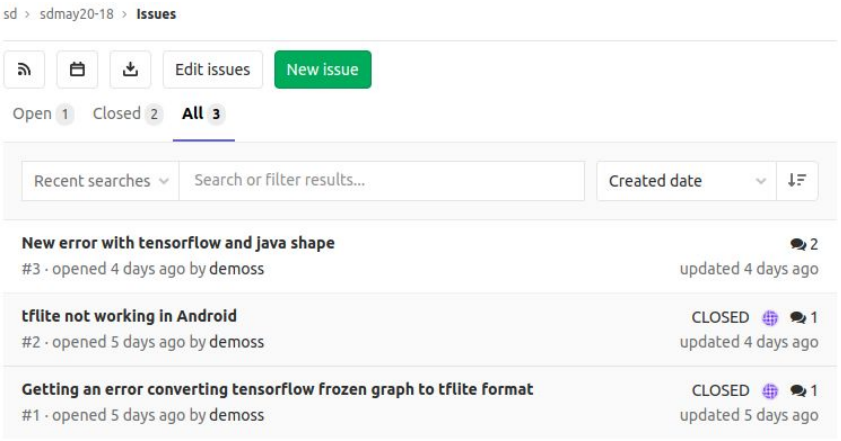


Figure 4. GitLab Issue Tracker

3.7 Expected Results and Validation

The desired outcome of our work is to create a 2 valid training models. Our first model will be intended for use directly on smartphones, and our second model will be a more heavyweight model which will perform more rigorous crack detection, including multiple levels of severity. Our model should be generalizable, meaning that the weights that we generate to perform our crack detection task should work during different times of the day and on pavement and cracks in different cities. Additionally our results from validation testing will help us determine if we have reached our goal.

4. Project Timeline, Estimated Resources, and Challenges

4.1 Project Timeline

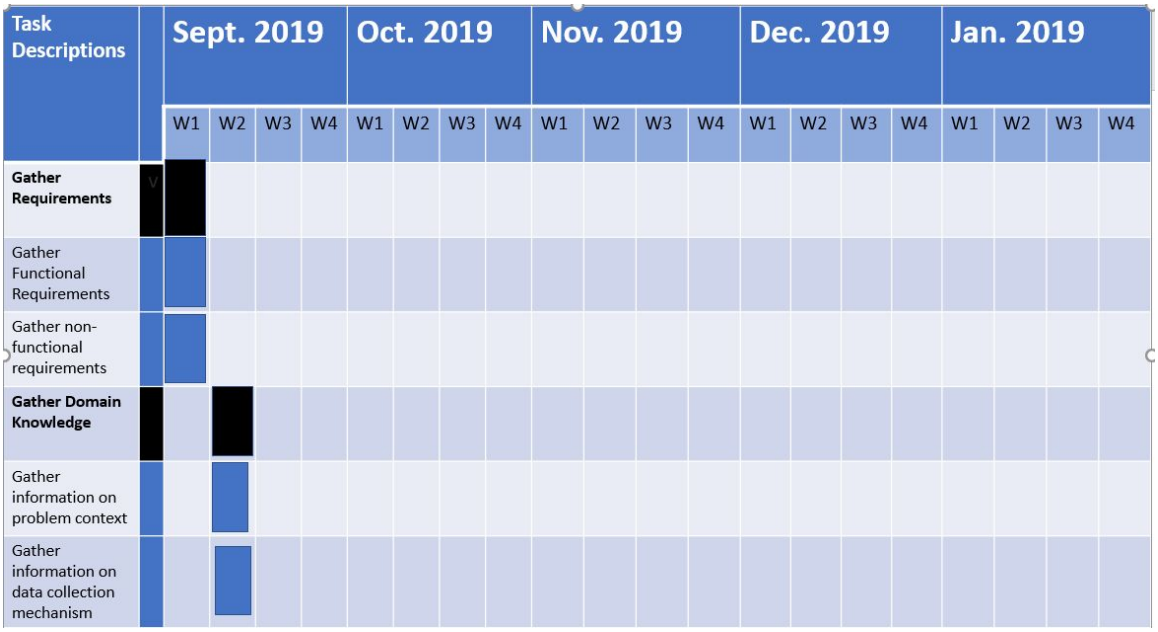


Figure 5. Gather Requirements and Gather Domain Knowledge Timeline.

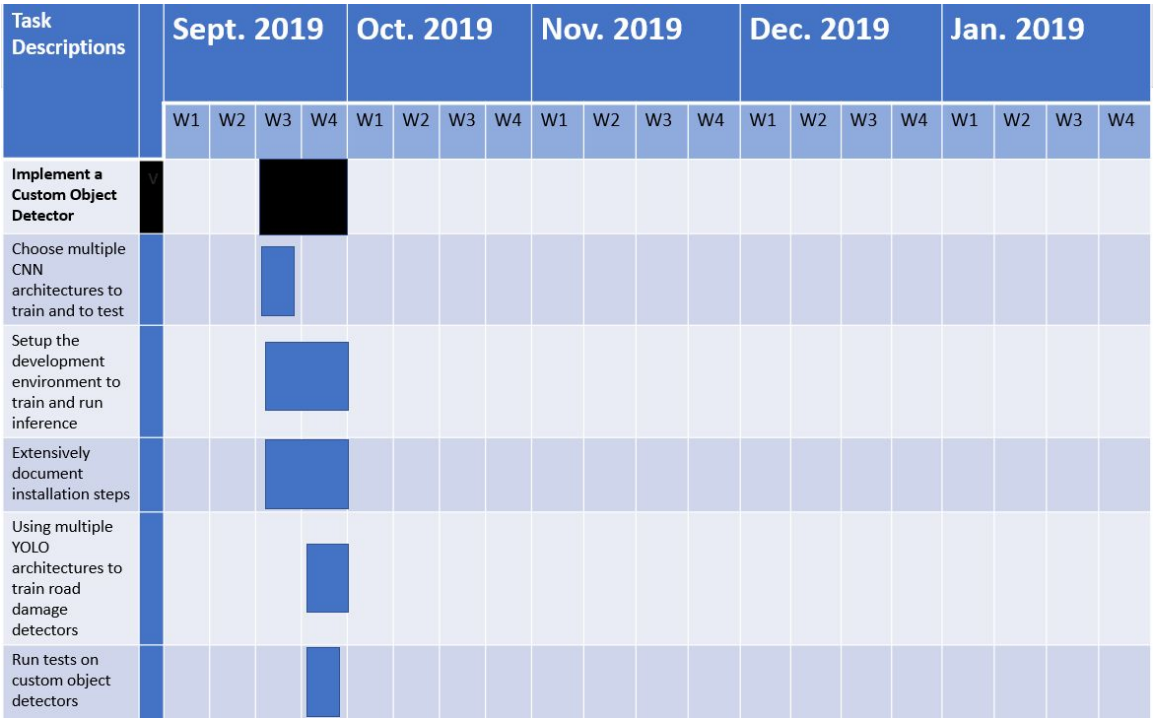


Figure 6. Implement a Custom Object Detector Timeline.

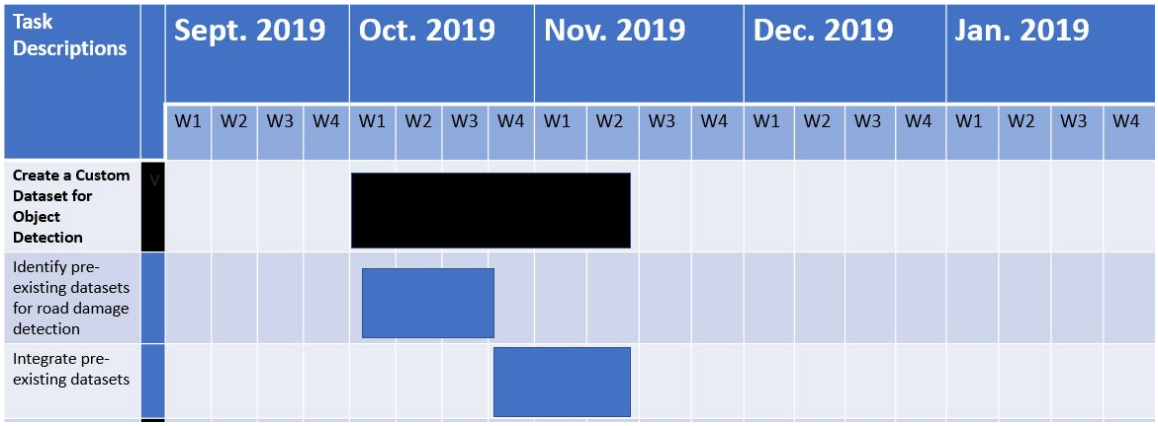


Figure 7. Create a Custom Dataset for Object Detection Timeline.

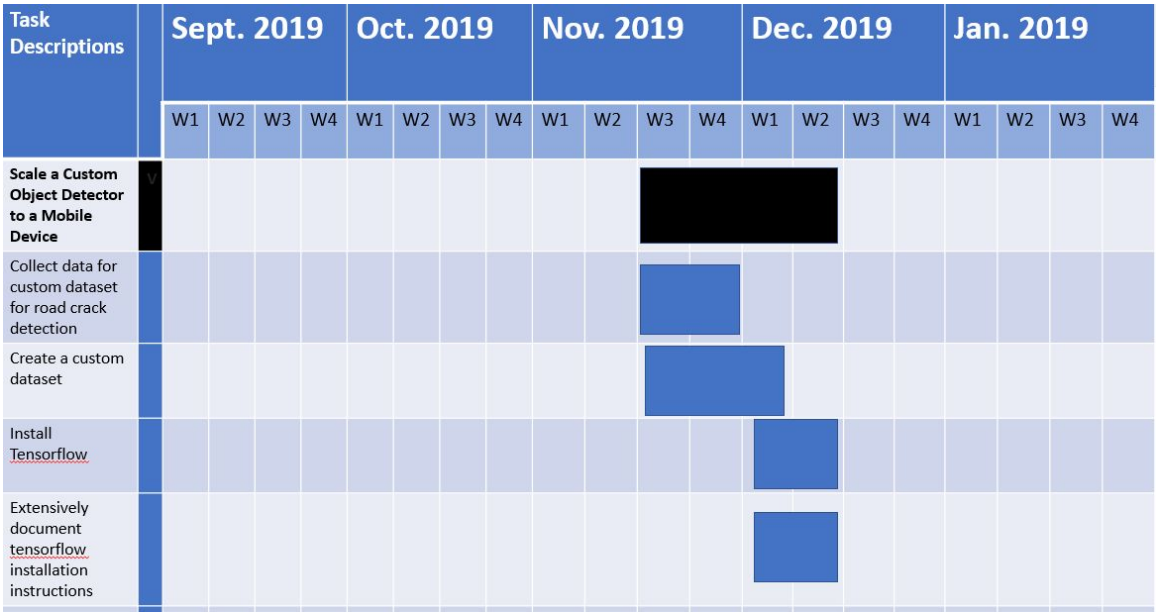


Figure 8. Scale a Custom Object Detector to a Mobile Device Timeline.

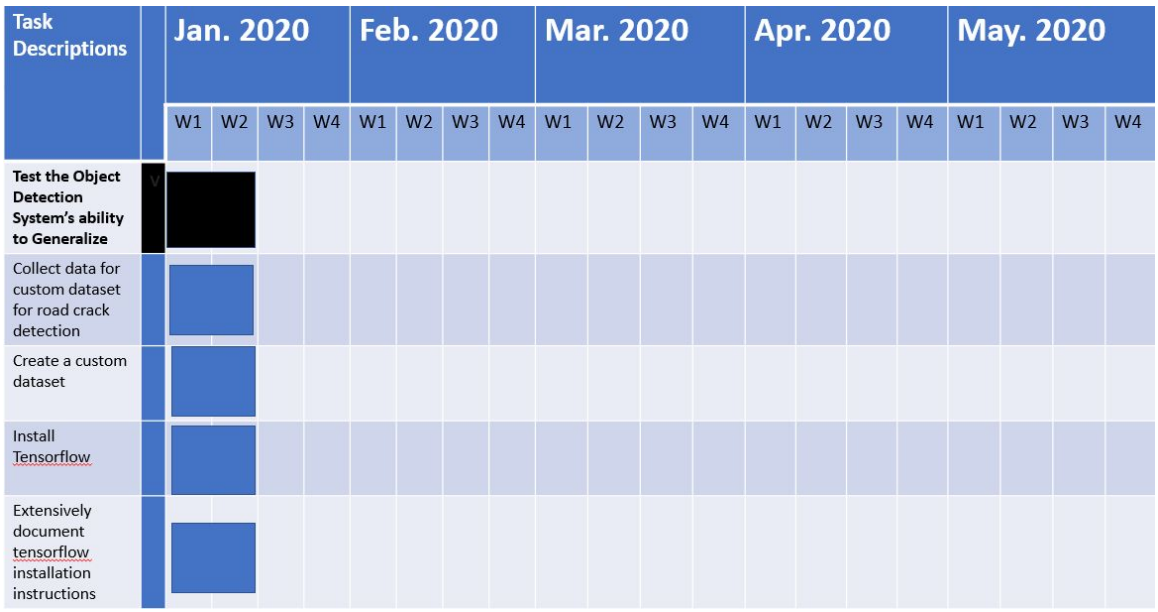


Figure 9. Test the Object Detection System's Ability to Generalize Timeline.

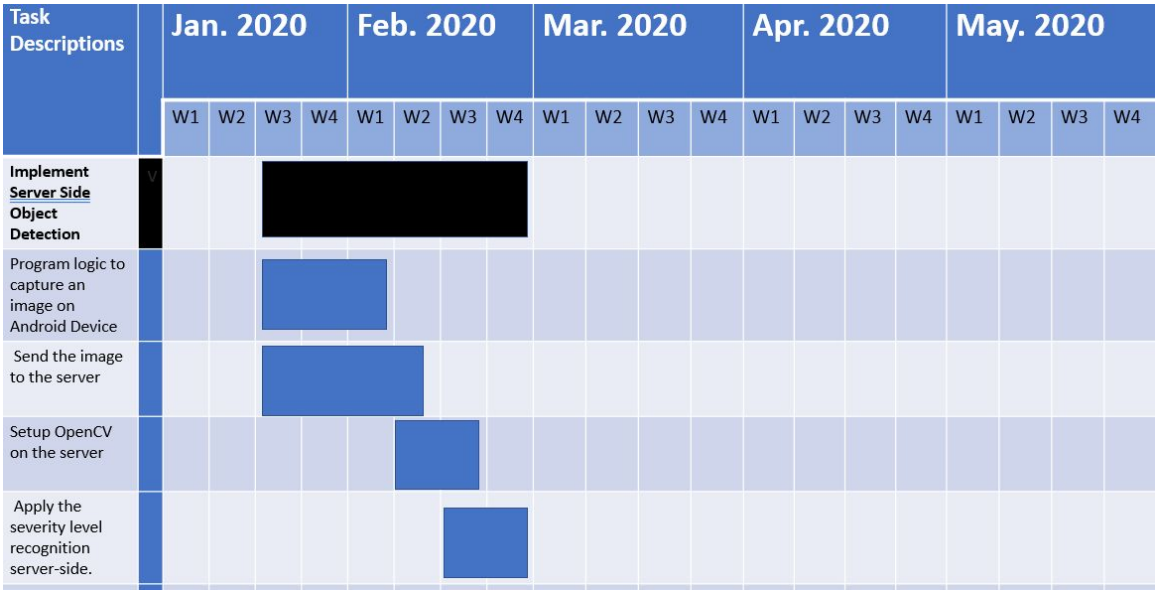


Figure 10. Implement Server Side Object Detection Timeline

Figure 11. Implement UX / UI Design Timeline.

Figure 12. System Level Testing Timeline.

4.2 Feasibility Assessment

The project will be a mobile application for Android smartphones that will automatically detect and classify cracks and potholes in the road. The phone will be mounted on the dashboard or windshield of the user's vehicle, with the camera facing down towards the road. The application will record and a video with geotags and timestamps recorded when a crack/pothole is detected. This information will be sent to a server, where it will subsequently be processed through the classification algorithm and returned with a list of the cracks/potholes, their location, and their classification.

We have quite a few foreseen challenges to this project. Since the user is mounting the device, there is a lot of different angles that the camera could be facing the road at. We must find a way to allow that kind of variation, or give them a strict method of mounting the phone. Additionally, since we will also we surveying highways, we need to be able to identify cracks at high speeds. Since phones can only record at 60fps, we may have to limit the max speed of the user.

4.3 Personnel Effort Requirements

Fall Timeline

Week	Task	Personnel Effort Requirements
Week 7 - 10/10	Cuda 9.0 +cudnn v7.3.0 and Darknet installed, update any documentation that needs to be created or updated regarding this process.	**5-6 hrs**
Week 8 - 10/17	OpenCV Version 4.0.0 installed, update any documentation for this install, should now be able to run and test training a Darknet model, identify key datasets for training	**3 hrs**
Week 9 - 10/24	Tensorflow r1.12 (larger build, may have more errors so extending this build and documentation step to span 2 weeks), choose criteria for labeling and begin labeling data for training	**5 hrs**
Week 10 - 10/31	Tensorflow r1.12, begin training data, training experiments: follow training steps from this document. Also test the accuracy difference in unprocessed and preprocessed datasets. Try	**7 hrs**

	preprocessing the data in multiple distinct ways.	
Week 11 - 11/7	Convert Tensorflow model to Tensorflow Lite model (challenging task, may need to switch to a different model if this has not been completed by this date)	**10 hrs**
Week 12 - 11/8	Integrate our converted models into Android app and begin testing our models to see which ones have the highest accuracy.	**10 hrs**
Week 13 - 11/15	Start training new models based on the prior noted tweaks	**4 hrs**
Week 14 - 11/21	Test these models	**2 hrs**
Week 15 - 11/28	Wrap up documentation and results, come up with plan for next semester	**3 hrs**
Week 16 - 12/1-1/13	Break	**None**

Table 3. Fall Timeline. Each date is a Thursday, which is when we meet with our advisor. The work is to be done in the week leading up to each Thursday. The personal effort requirements are also listed alongside each week's work.

Spring Timeline

Week 17 - 1/16	Begin developing frontend UI in android studio. Configure server and begin backend development.	**10 hrs**
Week 18 - 1/23	Begin incorporating detection into the frontend and classification into the backend.	**7 hrs**
Week 19, 20, 21 - 1/30-2/13	Begin testing algorithms on real world roads and making appropriate adjustments.	**4 hrs**
Week 22 - 2/20	Finish draft of user screens.	**12 hrs**
Week 23 - 2/27	Begin creating interaction between frontend and backend.	**10 hrs**
Week 24 - 3/5	Finalize interaction between frontend and backend.	**12 hrs**
Week 25, 26, 27, 28 **3/12-4/2**	Test application and make appropriate adjustments.	**40 hrs**
Week 29 - 4/9	Test out final draft of application.	**6 hrs**
Week 30 - 4/16	Make any final adjustments.	**10 hrs**

Week 31 - 4/23	Submit project and reflect on feedback.	**2 hrs**

Table 4. Spring Timeline. Each date is a Thursday, which is when we meet with our advisor. The work is to be done in the week leading up to each Thursday. The personal effort requirements are also listed alongside each week's work.

4.4 Other Resource Requirements

We will need YOLO, cuda, and Tensorflow lite for software requirements. We will also need a smartphone, a computer to host a server on, and a dashboard mount, which will be provided by our advisor.

4.5 Financial Requirements

We will require a laptop to use as our own project server. This way the project can be accessed by our client and advisor after we graduate.

5. Testing and Implementation

5.1 Interface Specifications

This project's outcome consists of implementing an algorithm to be used on a mobile phone mounted in a car. Therefore, the algorithm will run on a phone with a graphical interface with minimal user input once the user has initiated the program. i.e the user should be able to input all commands before starting the cracks/potholes detection. The interface should display a real-time video feed of identified transverse cracking, longitudinal cracking, and potholes. Thus, by using real-time data for the testing, we will improve the chances of the algorithm to detect issues that may occur on the user's end.

5.2 Algorithm Implementation

For our convolutional neural network architecture we chose SSD Mobilenet. SSD Mobilenet is an optimized version of the SSD architecture. It contains less filters, and

also leverages depth-wise separable convolution, which makes it importantly lightweight and able to detect fast on mobile devices while also detecting relatively accurately.

5.3 Android Application Implementation

The Android application was created using Android Studio and made to target API 27. This application was developed with both Java and Kotlin and utilizes Android's new CameraX library. To make inferences using the trained model, the app utilizes TensorFlow Lite as well as some of the functions provided by Google's Neural Networks API. Once detections are made, images and relevant information are saved to the device or sent to the server using HTTP requests aided by the Volley library. Upon opening the app, the user is prompted to log in using a simple login/account creation page. Once logged in, user interaction is simple and limited to a start/stop button which either starts or stops inferencing using the back-facing camera.

5.4 Backend Implementation

For the backend of this project we procured a Ubuntu VM from SSG in Coover. On this VM we operate our Node.js server which has routes for all HTTP requests for the project as well as the MySQL database. HTTP requests are sent to the Node.js server, which then executes functions to format and store/retrieve the data within the database. Images sent to the server are stored on the server filesystem, with references to the image name within the database.

5.5 Web Portal Implementation

The web portal was implemented using React.js, JavaScript, HTML, and CSS with an account creation page, login page, and dashboard. Both the account creation and login pages are simple, standard, and intuitive user interfaces. The dashboard consists of two vertical panes. Left consists of a list of list items that include unique identifiers for cracks/potholes as well as detected types. The list is sortable based on type. When a crack is selected from the left pane its details are displayed within the right pane. Details included are location, date, time, type, image of the crack, and a Google Maps pin of the location on a map. The map and pin are included using the Google Maps API and GPS location sent from the Android application.

5.6 Functional Testing

To verify that the final product meets the quality expectations and requirements specifications, an actual driving test will be performed to locate defects in the algorithm. During this test, we will drive (in normal weather conditions) in certain areas and collect

enough data which we will in turn use to assess the actual outcome compare to what is expected. Nevertheless, this testing mechanism does not guarantee that all defects will be identified. Thus, with the outcome, we will decide whether to improve our algorithm and fix the issues (if severe) or whether to consider that the algorithm minimum requirements have been met.

To achieve functional testing, we will identify functions that the algorithm is expected to perform. Then create input (i.e. feeding images/videos) data based on the function's specifications and determine the output based on these specifications. Finally, we will execute test cases and compare the actual and expected outputs.

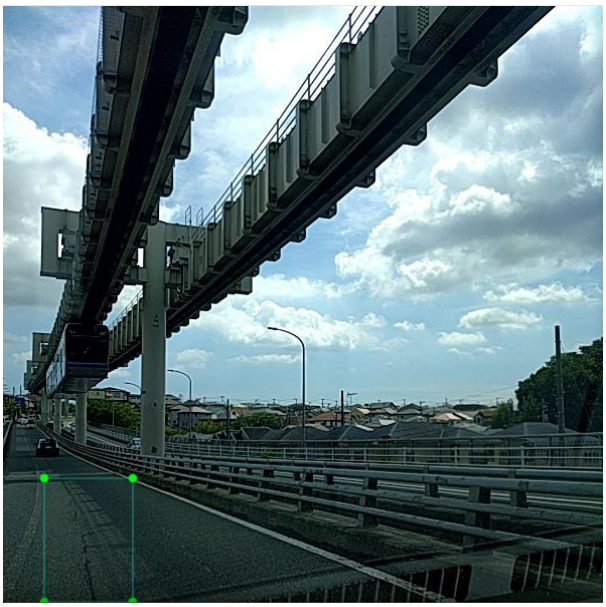


Figure 13. Ground truth label before training yolo v2 weights

Figure 14. Bounding box generated from inference.

5.7 Non-Functional Testing

While functional testing will test the functionality of the algorithm, non-functional testing is also very important and should be taken into consideration right from the inception of the algorithm. Hence, the user should be able to mount the mobile phone in the car prior to its usage and shouldn't need to have prior knowledge about how to use the application.

5.8 Process

As indicated in section 2, there are three major processes to select from: a mobile application with all the detection features included; a mobile application that transfer all the data collected to a server which does all processing; or a spit of detection features between the mobile app and the server. To be more efficient, we used a combination of those methods. The following illustrates how we tested those processes.

To test the mobile application, we looked at the images that were sent on the server by the app and verified that each of them depicts the types of cracks/potholes we were expected. In addition, we made sure that for each of those images, the location coordinates also match the values in our database. A similar approach was done to test the Node.js server, except the images from the mobile application are not processed prior to their transfer to the server. i.e, the server does all the processing. Finally, for the SQL Database, we test the output by using a large dataset of stored images, locations, and severity classifications.

5.9 Results

So far, we have a training environment setup and configured with CUDA and cuDNN module. We have tested multiple machine learning frameworks such as Keras and TensorFlow. By using tensorflow to start the training of some models for object detection, we have successfully trained a custom model, identified a sample dataset to test the framework on, and have a good understanding on how to label images in labelImg. In addition we have a custom configuration of OpenCV to build with CUDA and cuDNN modules among other custom modules. Finally, we have visualized results of the model using OpenCV 4 which drew the bounding boxes around the object detection predictions.

Figure 15. Object detection performs system transverse and longitudinal crack detection

Figure 16. Object detection system detects potholes.

6. Closing Material

6.1 Conclusion

To sum up, the goal of this project was to develop an algorithm that can analyze the images/videos taken by a phone camera and identify the types and severity levels of cracks found. The best plan of action to achieve this goal was to automate this analysis using feeds recorded by a smartphone mounted on the windshield of a car; this technique coupled with GPS coordinates proved to help engineers locate the cracking position and develop a pavement maintenance/rehabilitation plan accordingly.

6.2 References

[1] H. Maeda, Y. Sekimoto, T. Seto, T. Kashiyama, and H. Omata, “Road Damage Detection and Classification Using Deep Neural Networks with Smartphone Images,” Computer-Aided Civil and Infrastructure Engineering, vol. 33, no. 12, pp. 1127–1141, 2018.

[2]A. Alfarrarjeh, D. Trivedi, S. H. Kim, and C. Shahabi, “A Deep Learning Approach for Road Damage Detection from Smartphone Images,” *2018 IEEE International Conference on Big Data (Big Data)*, 2018.

[3] W. Wang, B. Wu, S. Yang, and Z. Wang, “Road Damage Detection and Classification with Faster R-CNN,” *2018 IEEE International Conference on Big Data (Big Data)*, 2018.

[4] Mei, Qipei & Gül, Mustafa. (2019). A Conditional Wasserstein Generative Adversarial Network for Pixel-level Crack Detection using Video Extracted Images.

[5] K. Gopalakrishnan, “Deep Learning in Data-Driven Pavement Image Analysis and Automated Distress Detection: A Review,” *Data*, vol. 3, no. 3, p. 28, 2018.

[6] Web.stanford.edu. (2019). [online] Available at: https://web.stanford.edu/class/ee368/Android/Tutorial-3.pdf [Accessed 4 Oct. 2019].

[7] Yang, F., Zhang, L., Yu, S., Prokhorov, D., Mei, X. and Ling, H. (2019). Feature Pyramid and Hierarchical Boosting Network for Pavement Crack Detection. *IEEE Transactions on Intelligent Transportation Systems*, pp.1-11.

[8] Cubero-Fernandez, A., Rodriguez-Lozano, F.J., Villatoro, R. et al. Efficient pavement crack detection and classification. J Image Video Proc. 2017, 39 (2017).

[9] Y Huang, B Xu, Automatic inspection of pavement cracking distress. J. Electron. Imaging. 15(1), 013–017 (2006).

[10] L Li, L Sun, G Ning, S Tan, Automatic pavement crack recognition based on Bp neural network. PROMET-Traffic Transp. 26(1), 11–22 (2014).

[11] Weixing W., et al. Pavement crack image acquisition methods and crack extraction algorithms: A review, Journal of Traffic and Transportation Engineering (English Edition), vol 6, no. 6, 2019.

6.3 Appendices

APPENDIX A: OPERATION MANUAL

APPENDIX B: ALTERNATIVE DESIGNS

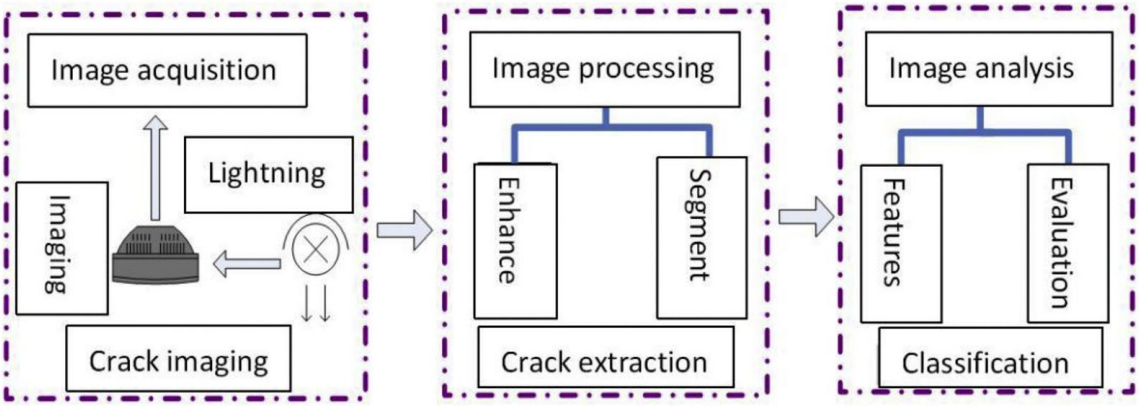


Figure 17: Diagram of Crack Detection System

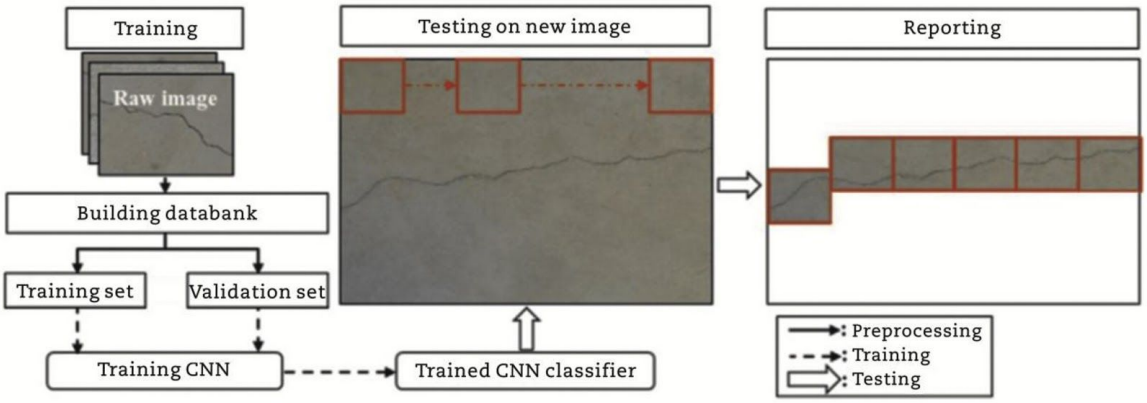


Figure 18: Deep Learning Framework on Convolutional Neural Network

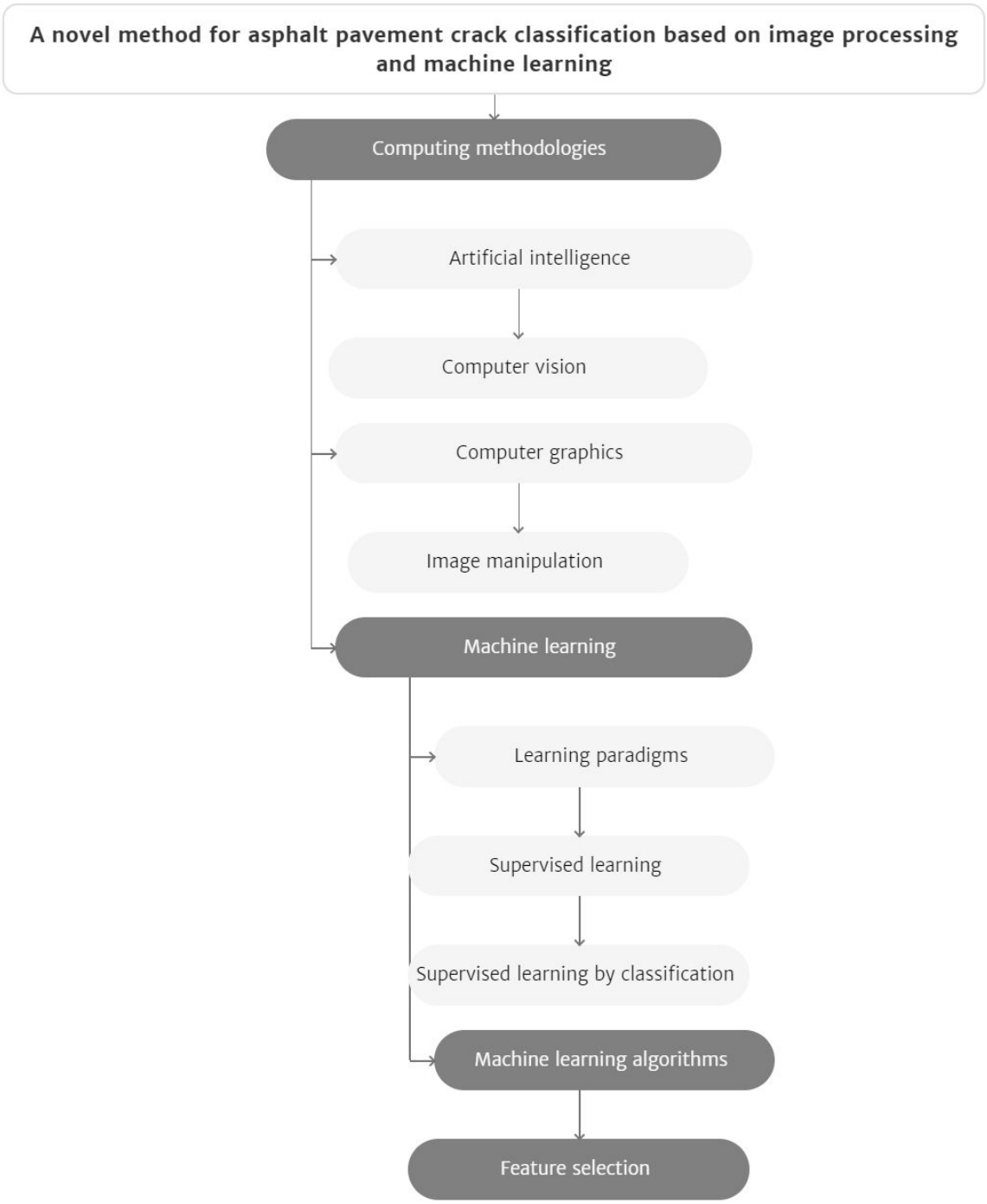


Figure 19: Crack Classification Based on Image Processing and Machine Learning

APPENDIX C: OTHER CONSIDERATIONS

Image-based techniques are fundamental in pavement crack detection. Various approaches have been proposed to detect cracks. Below is a brief overview of other

methods that should be taken in consideration. These methods can be combined to improve the overall detection of cracks.

Huang and Xu [9] describe in their work how they implement a complex system to capture images of the pavement to detect cracks. Once the image is captured, three main steps are performed: divide the image in grid cells of 8×8 pixels and classify each cell into a non-crack or crack cell. After that, crack cells are verified, comparing each one to their neighbors and, finally, crack clusters are connected to form the actual final cracks. Based on the starting and ending coordinates, the crack is finally classified as longitudinal or transverse.

Li et al. [10] take advantage of neural networks to detect cracks. Before the learning and classification process, a preprocessing step is performed. First, the image background is corrected, by making it more uniform. Then, in order to transform the image into a smoother one, a Gaussian smoothing is used. Finally, a histogram transformation is applied to highlight the crack. After the preprocessing, several techniques are used to split the image into smaller subimages. The background is removed and each sub-image is classified depending on whether it contains a crack or not. Only the images that contain cracks are selected and two different neural networks are used to classify these images. The first one classifies whether the crack has linear (either transverse or longitudinal) or alligator form. After that, the second one classifies between longitudinal and transverse cracks.

Once the images are captured, Cubero et al. [8] describe how several processes are applied in order to extract the main characteristics for emphasizing the cracks. After image preprocessing, a decision tree heuristic algorithm is applied to finally classify the image. This process produces an average of 88% of success detecting cracks and an 80% of success detecting the type of the crack. This method can be implemented in a vehicle traveling as fast as 130 kmh or 81 mph.