# Snapcrack

Senior Design Spring '20 Group 18
Client: Bo Yang, Advisor: Halil Ceylan
Maggie Dalton, Akira Demoss, Modeste Kenne, Nik Thota
Website: http://sdmay20-18.sd.ece.iastate.edu

## Problem

Finding and classifying cracks and potholes found in roads is dangerous and time consuming for researchers. It often requires them to park on the side of busy roads and leave their vehicles to gather data for each crack.

## Intended Users & Uses

Researchers gathering data on cracks and potholes found in Iowan roads. It will be used while they drive to compile data for their research projects.

## Solution

Eliminate the need for researchers to leave their vehicle by detecting, classifying, and storing images of cracks and potholes using an Android application. Detected cracks/potholes are stored in a database and made accessible through a web portal.

## Operating Environment

- Android application run on an Android phone running Android API 25+
- Phone mounted on the dashboard of the car running the application in the foreground
- Operational during the daytime without adverse weather (rain/snow/etc.)

## Engineering Standards & Design Practices

- Standardized Tooling and Technologies
  - Agreed on tools and technologies
- Standardized Coding Style
  - Defined conventions for coding
- Process Standards
  - Agreed on development steps
- Code Quality Control
  - Evaluated to make sure code is understandable

## Functional Requirements

- The application can detect longitudinal cracks, transverse cracks, and potholes found in roads
- Images of detected cracks/potholes will be saved
- Location, date, time, and type of detected cracks/potholes will be stored in a database

## Non-Functional Requirements

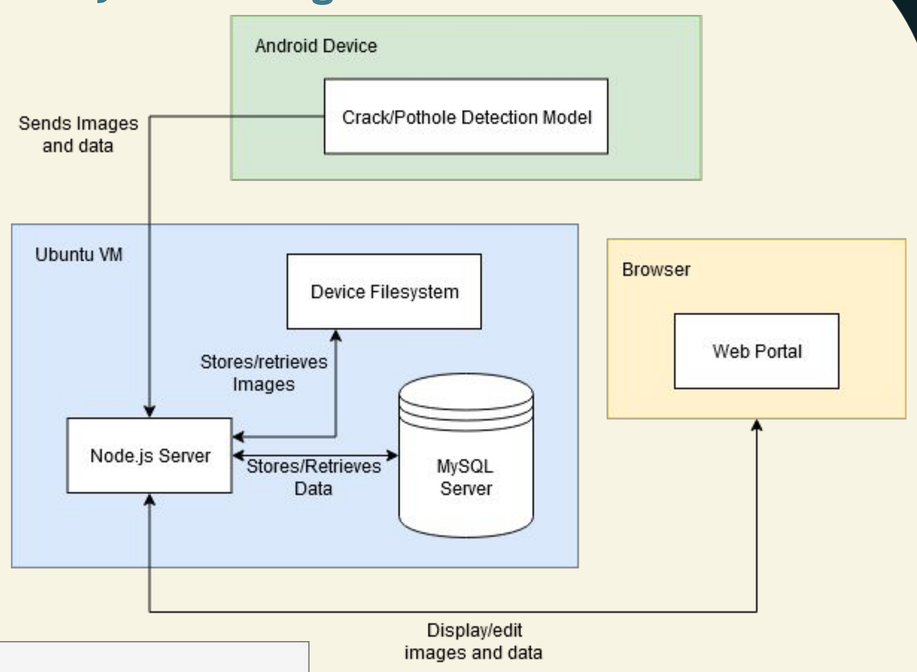
- The application should be capable of making detections on multiple road types (composite, asphalt, etc.)
- The app shall be capable of making detections at a variety of speeds
- Detections made should comply with the LTPP distress manual
- The user shall be capable of viewing past detections along with relevant data

## Technical Details

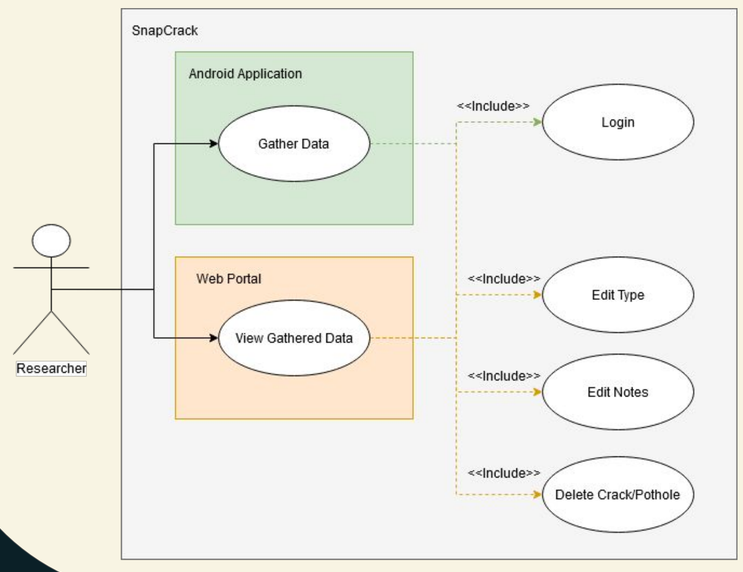
- Backend
  - Ubuntu VM
  - Node.js server
  - MySQL Database
- Web Portal
  - React
  - JavaScript/HTML/CSS
  - Google Maps API
- Model
  - TensorFlow
  - OpenCV
  - CUDA capable GPU
- Android Application
  - Java/Kotlin
  - TensorFlow Lite

## System Design

Component Diagram




Use Case Diagram




## Testing

**Datasets**

Training data: 1405 images, 3055 labels
Testing data: 350 images, 756 labels

**Mean Average Precision (mAP) Performance Benchmark**

Intersection over Union (IoU) = Area of intersection / Area of union
True positive (TP): IoU > 0.5
False positive (FP) IoU <= 0.5
Precision = TP / (TP + FP)
Mean Average Precision (mAP) = $\frac{1}{|classes|} \sum_{c \in classes} \frac{TP(c)}{TP(c) + FP(c)}$