SnapCrack

Akira Demoss, Maggie Dalton, Modeste Kenne, Nik Thota

sdmay20-18

Client: Bo Yang

Adviser: Halil Ceylan

Team email: sdmay20-18@iastate.edu

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

Revised: 10/6/2019 (v1)

Executive Summary

Development Standards & Practices Used

List all standard circuits, hardware, software practices used in this project. List all the Engineering standards that apply to this project that were considered.

Summary of Requirements

List all requirements as bullet points in brief.

Applicable Courses from Iowa State University Curriculum

List all Iowa State University courses whose contents were applicable to your project.

- CPRE 388 (Embedded Systems II: Mobile Platforms)
- COM S 309 (Software Development Practices)

- COM S 311 (Algorithm Analysis & Design)
- COM S 363 (Database Management)

New Skills/Knowledge acquired that was not taught in courses

List all new skills/knowledge that your team acquired which was not part of your Iowa State curriculum in order to complete this project.

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

Table of Contents

List of figures/tables/symbols/definitions (This should be the similar to the project plan)

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

The project is trying to minimize the risk that researchers encounter when evaluating roads for road maintenance and increase the speed at which they can evaluate roads. Currently, researchers must park alongside roads and measure cracks/potholes by hand to determine the severity of the damage. This measurement is a vital part of determining where to focus research and gather data. This process is both slow and dangerous; researchers are often standing alongside lanes of high speed traffic.

Our approach aims to remove the need for people to stop their vehicle and gauge damage severity. This project will be capable of identifying target crack types and potholes and classify them while the users are driving. Identified areas will be stored in a database with their location and severity for later use.

This is planned to be accomplished with a mobile device running a mobile application running a trained YOLO algorithm 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 for classification. The server will run the image through a classification algorithm to determine the severity of the pothole/crack and then store the classification, location, and other important data in a database for use by the users.

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 capable of interacting 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 Requirements

- Easy to use
    - Simple UI design that requires no prior knowledge to operate

    - Minimal setup required to begin use
    - No additional actions required from the user once started until the user chooses to end use of the application
- Low cost
    - No additional hardware outside of the mounting device
    - Cost-effective server and database management
- Able to identify/classify on diverse roads
    - Ability to function at a variety of speeds (10mph - 70mph)
    - Ability to function with different road types (composite/asphalt/etc.)
    - Ability to detect and classify specified crack types/potholes using criteria from the LTPP distress manual
        - Transverse Cracks
        - Longitudinal Cracks
        - Potholes
    - Saves the location of detected cracks/potholes

1.5 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.

1.6 Assumptions and Limitations

Assumptions

- The mobile device will be mounted using a third party mounting device on either the dashboard or windshield
- The application will be started prior to the user driving, requiring no interference until the user has finished evaluating their target area
- The product will not be used outside of the United States
- There will be 10 or fewer concurrent users sending/receiving data from the server at a given time
    - Primary focus as defined by the client is on creating a functioning and accurate algorithm and less on supporting multiple users
- Data obtained from the application and any processing of the data will be stored in a centralized database
    - We don't currently foresee different organizations using this application simultaneously

Limitations

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

- The application should be simple to use, requiring no prior knowledge of machine learning, LTPP crack classification, or any other prior knowledge to operate
- The classifications will be categorized using criteria from the LTPP distress manual

1.7 Expected End Product and Deliverables

The deliverables include an Android application, Node.js server, and SQL database.

The Android application shall be free and available on the Google Play Store. It will require no additional downloads 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 capturing images of roads while the user is driving and sending them to the delivered server.

The server and database will be made available to the client with little to no additional action required to operate. It will be simple to start/terminate the server and no additional action should be required in order to utilize the application in its full capacity. They will operate exclusively with each other and the delivered Android application. The database will be capable of storing relevant data to the classification of cracks/potholes as specified by the client.

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

If time permits, an iOS version of the Android application may also be delivered. This may or may not function to the same capacity of the Android version, but would retain the base functionality of detecting cracks/potholes and communicating with the server.

All deliverables outlined in this section are expected to be made available at the end of the Spring 2020 semester upon the completion of the CPRE 492 course by the project members.

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, and breaking the detection and classification up between the mobile app and the server. 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 and classification server side would be extremely accurate but require sending video to the server, and thus be extremely data intensive and requires either enormous storage or cellular data. Lastly, a blended approach would allow us to detect cracks/potholes at a relatively quick pace while maintaining accuracy in the more challenging area of classification with less data.

Our approach

While we recognize the value in the other two approaches, we settled on a blended application. Our current proposed design is as follows:

- Mobile application

    - Running a trained YOLOv3-tiny model using TensorFlowLite for detecting potholes and cracks
    - Captures images when a potential crack or pothole is detected
    - Captures the location of where the image was taken
    - Sends the image and relevant data to the server either as detections are made or in a batch at user request
- Node.js server
    - Receives the image and relevant data from the Mobile application
    - Stores the base image and location in the database
    - Sends the image through a trained classification model for severity classification
    - Receives the output from the classification algorithm and stores the result with the input image in the database
- SQL Database
    - Stores the captured and unedited image, location, severity classification, and a unique identifier for each detected crack/pothole

We feel that this combination of technologies will let us have a short inference time to detect cracks/potholes and maximum accuracy for classification of severity. One requirement is that the app functions at different speeds, and thus it needs to work quickly enough to scan roads at high speeds.

2.2 Design Analysis

With this design approach we are able to combine the strengths of the other two options. We believe that it 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. In addition, doing the classification server-side will increase the accuracy of classification. Mobile devices have limited hardware and therefore take longer to make inferences. If both detection and classification were done in-app it is likely that it will not be capable of performing at moderate to high speeds.

It is also worth noting, however, that this design comes with some concerns. While we are hopeful that we can make timely inferences, it may not be possible to detect cracks/potholes at high speeds (where the demand for this project is the greatest). Limitations in mobile hardware with both the camera and processing power may inhibit the speed at which the model can make detections.

Initially, we were apprehensive of this blended approach being attainable, but as we continue to research and implement prototypes it is increasingly proving to be a valid option. We have found an offshoot of TensorFlow namedTensorFlowLite that is optimized for mobile. After working through some examples of TensorFlowLite YOLO detection, we feel that with the right adjustments the application would be able to inference quickly. Although we do not have a prototype with cracks/potholes yet, we have tested it using pre-trained models for common items and have found that it inferences with an average of around 50ms per inference.

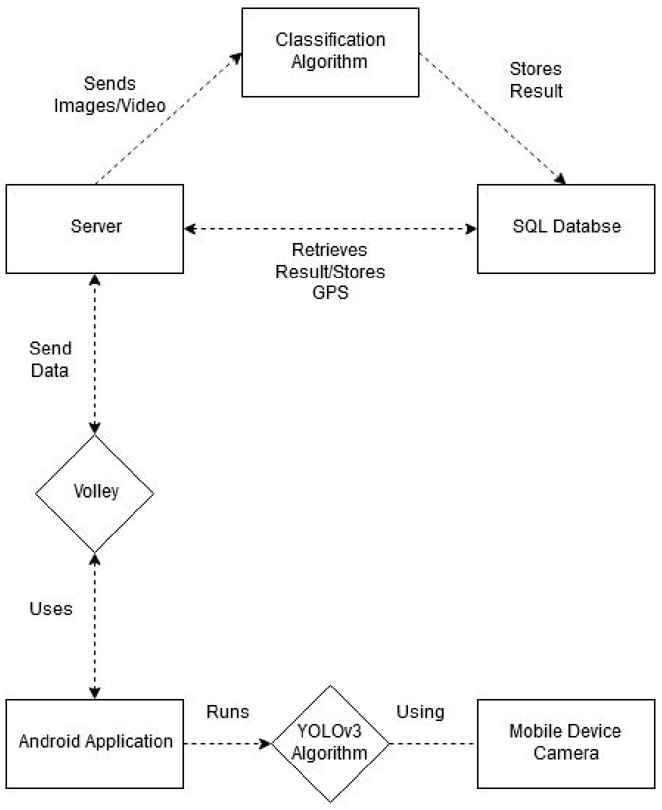
Moving forward, we will likely stick to this blended approach but how successful a prototype for cracks/roads performs will influence what type of data we send to the server. In our opinion, the optimal solution would be if we could inference then snap an image and only send images with detected cracks/potholes to the server. This would minimize the required data for both detection and classification. It is currently unclear, however, if the action of inference and taking a picture would be quick enough to effectively scan the road while driving without creating blindspots. In the event that images would be too slow, we plan to take video and record timestamps where detection has occurred along with the GPS location of the detection.

2.3 Development Process

We are following a Kanban Agile approach to this project. The components of this project (i.e. mobile application, server, etc.) have been divided into sections (i.e. YOLOv3 model, Android UI, API, etc.), and each section has been divided further into small tasks (i.e. label dataset, add model to Android, etc.) using cards on Trello. Tasks on each card are usually discussed at the time they're added to the board. Each card is kept in the backlog of the Trello board until members finish their current in-progress tasks, at which time they can decide upon which card they work on next from the backlog. The majority of the time, team members choose cards based on their background and areas of the project they're most interested in working on. Generally, team members only have 1-2 active in-progress cards and 0-1 issue cards and complete 1-2 cards per week. If there is a card that needs to be completed soon, but hasn't been picked up voluntarily, the team will discuss the card during the next meeting. During that meeting the card will be assigned to a member based on their skills, the current workload of the other members, and how urgently the card needs to be completed.

We've had challenges meeting due to diverse schedules, but discussing and documenting cards when they're created and adopting a Kanban approach has let us make the most of the time we meet. Members have ownership over their cards and aren't dependent on a team meetings to continue making progress. Each person can take on or complete cards at will. The limit on cards and documentation on Trello also make it clear what each person is currently working on. Overall, this approach has worked for us so far.

2.4 Design Plan



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 detection. In reference [1] and [2], simple bounding box detectors are used to detect various road anomalies. In reference [3], segmentation is used to detect road anomalies. Finally, in reference [4] an alternative approach to crack detection using a conditional Wasserstein Generative Adversarial Network (WcGAN) is described. 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.

In [1], SSD Mobilenet is used for detection. While this algorithm is more accurate, the detection speed is slow. In [2], they reference using YOLO, however they provide no evidence of having an actual working model in a smartphone. 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. The lighter weight algorithm utilizes less convolutional layers and also uses max pooling layers as opposed to residual layers. Because these layers also determine the feature maps, it logically makes sense that this will reduce the accuracy, while giving us a boost in detection speed. This will allow us to run the algorithm in real time which is favorable for our project because we hypothesize the trade off in accuracy for real-time performance will improve the number of detected objects. For example, a highly accurate detection algorithm which is only capable of running at 3 frames per second is not useful for detecting cracks in a vehicle that is moving at 50 miles per hour. This is because by the time the detector has run, it may have already missed dozens of cracks due the time spent processing the data. With our proposed approach we will be able to easily map the gps coordinates to road anomalies picked up by our algorithm. In approach [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 in [4], an approach using a conditional Wasserstein generative adversarial network (WcGAN) is used. While very accurate, the disadvantage is the large computational costs e.g. method were tested on a desktop with Intel 8700k CPU, 32GB memory and Nvidia Titan V GPU with 5120 CUDA cores. If time permits, we may use use a GAN to generate synthetic training data, however this is currently a stretch goal.

3.2 Technology Considerations

For technology we are limited to what we personally own. Access to laptops with administer rights were thankfully provided to us by the SSG, these laptops have CUDA capable gpus which will allow us to all work on the algorithm, however because our gpus are limited in computational power, we will not be able to train quickly. Despite this we are still confident we will be able to develop a valuable research product.

3.3 Task Decomposition

Task 1.) Set up the development environment and document setup steps so that work can be reproduced by our client. Because our client is a researcher and his background is in another expertise, it will be valuable for our client to have a way to reproduce our results. This task involves installing Cuda 9.0 +cudnn v7.3.0 and Darknet, OpenCV Version 4.0.0 installed, and Tensorflow r1.12 and updating any documentation that needs to be updated regarding these installation steps. Additionally, we will need to test to make sure that everything is installed correctly. We can test this by running scripts that have been made previously and verifying that the build is working.

Task 2.) Run the training algorithm on "toy" datasets to gain a basic understanding of how model configuration and training work. For this task we will find pre-existing datasets that follow deep learning best practices and use these to learn the basics of configuring a model for training. After we are comfortable with this, we will move on to identifying datasets that will work for our specific task.

Task 3.) Training a model for basic anomaly detection. The two types of road anomalies we will need to detect are cracks and potholes. We will write a basic detection algorithm that is fast, lightweight, and will run on a mobile phone so that we are able to trace the precise gps coordinates of our detected anomalies.

Task 4.) Once we have a basic detection algorithm written, we will work to refine this algorithm. There are several experiments that we have in mind regarding refining our algorithm. The first step will be to test and see if preprocessing our data will give us more accurate detection results. In this task we will need to create a realistic test dataset and apply various preprocessing techniques to our existing dataset. Additionally to refine our algorithm we will experiment with tweaking important parameters in our neural network. The two most important parameters that we will work on tweaking include the learning rate, and regularization.

3.4 Possible Risks And Risk Management

There are many challenges for this project. We are working to achieve results on a solution that leverages deep learning with limited computational resources outside of the laptops we were able to check out from the SSG. Some papers cite using a 32GB graphics card with state of the art CPU, while we will be working on a 2GB graphics card for all of our training experiments. Additionally there is a shortage of faculty with deep learning expertise and no one we have been able to identify for guidance on our approach additionally none of our team members have had formal coursework in the area of deep learning. This means that along with completing tasks it is necessary for us to also educate ourselves and each other on findings and how things work to make sure that our methods and approach to our project are logical and follow best practices in alignment with the field of deep learning. Because there still remain areas of the project in which we "don't know what we don't know", it is likely we may have periods of time where we need to spend more time educating ourselves than completing physical tasks and objectives in order to come up with a logical and effective solution that will be able to be further developed. Finally, inherently deep learning requires a complex technology stack and

3.5 Project Proposed Milestones and Evaluation Criteria

Task 1 Milestones

- Cuda 9.0 + cudnn v7.3.0 and Darknet installed, update any documentation that needs to be created or updated regarding this process.
- OpenCV Version 4.0.0 installed, updated documentation for this install, able to run and test training a Darknet model, identify key datasets for training.

- Tensorflow r1.12 built, and criteria for labeling data for training.

Task 2 Milestones

- “toy” dataset identified. This dataset should have positive and negative examples, and include a script which divides the dataset into training, testing, and validating data respectively.
- Weights trained from a “toy” dataset to validate that software has been properly configured.
- Identified key prospective datasets for our application and evaluate if it makes sense to create our own dataset, or if adequate pre-existing public datasets exist.

Task 3 Milestones

- Identify and implement strategies for preprocessing data and data augmentation to be applied prior to training and improve detection accuracy this way.
- Tweak model configuration parameters which result in improved detection accuracy.

3.6 Project Tracking Procedures

Our group is using a Trello board to track work that is in progress, on the backlog, and completed.

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

Based off of Task list

Timeline:

Fall Semester:

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.

Week 7 - 10/10 (5-6 hrs): Cuda 9.0 +cudnn v7.3.0 and Darknet installed, update any documentation that needs to be created or updated regarding this process.

Week 8 - 10/17 (3 hrs): 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

Week 9 - 10/24 (5 hrs): 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

Week 10 - 10/31 (7 hrs): 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 preprocessing the data in multiple distinct ways.

- Train with unprocessed dataset
- Train with preprocessed data

Week 11 - 11/7 (10 hrs): 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)

Week 12 - 11/8 (10 hrs): Integrate our converted models into Android app and begin testing our models to see which ones have the highest accuracy.

- Choose models that have the highest accuracy and further tweak these e.g. which preprocessing techniques improved accuracy, try combinations of these.
- When it comes to training neural networks and tweaking parameters, the most important parameter is the Learning Rate. Look into ways to optimize this parameter and continue training models
- Next most important parameter is Regularization. Look further into this.

Week 13 - 11/15 (4 hrs): Start training new models based on the prior noted tweaks

Week 14 - 11/21 (2 hrs): Test these models

Week 15 - 11/28 (3 hrs): Wrap up documentation and results, come up with plan for next semester

Week 16 - 12/1-1/13: Break

Spring Semester:

Week 17 - 1/16 (10 hrs): Begin developing frontend UI in android studio. Configure server and begin backend development.

Week 18 - 1/23 (7 hrs): Begin incorporating detection into the frontend and classification into the backend.

Week 19, 20, 21 - 1/30-2/13 (4 hrs): Begin testing algorithms on real world roads and making appropriate adjustments.

Week 22 - 2/20: Finish draft of user screens.

Week 23 - 2/27 :(10 hrs): Begin creating interaction between frontend and backend.

Week 24 - 3/5 (12 hrs): Finalize interaction between frontend and backend.

Week 25, 26, 27, 28 (40 hrs) - 3/12-4/2: Test application and make appropriate adjustments.

Week 29 - 4/9 (6 hrs): Test out final draft of application.

Week 30 - 4/16 (10 hrs): Make any final adjustments.

Week 31 - 4/23 (2 hrs): Submit project and reflect on feedback.

This is our proposed

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

Refer to section 4.1 for the personnel effort requirements.

4.4 Other Resource Requirements

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

4.5 Financial Requirements

Our project has no financial requirements.

5. Testing and Implementation

5.1 Interface Specifications

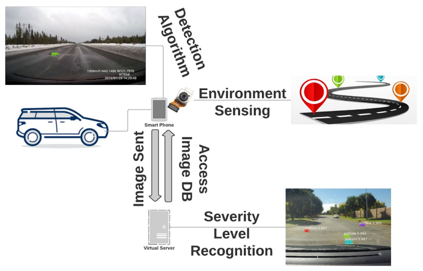
This project's outcome consists of implementing an algorithm to be used on a mobile phone mounted in a car. Therefore, the algorithm will run on a phone with a graphical interface with minimal user input once the user has initiated the program. i.e the user should be able to input all commands before starting the cracks/potholes detection. The interface should display a real-time video feed of identified transverse cracking, longitudinal cracking, and potholes. Once identified, it will evaluate the severity of the cracking using the LTPP distress manual and store the processed data. 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 Hardware and software

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 we the algorithm minimum requirements have been met.

5.3 Functional Testing

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.



5.4 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.

5.5 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.

5.6 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 Darknet backend 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 labelIng. 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. Hence, we are currently in the process of modeling and simulating our algorithm. Although we expect to have a prototype by the end of this semester (per the project specifications), the algorithm will be finalized to meet our client’s expectations by the end of next semester.

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.

6.3 Appendices

Coming soon...