LIVESPOT

May18-37	
Client	Iowa State Parking Division
Advisor	Dr. Ahmed Kamal
Team Members – Roles	Derrick Lockwood – Project Manager Donavan Brooks – Backend Lead Riley Snyder - Webmaster Mason Schreck – Communications Lead John Ingwersen – Mobile Master Joe Krajcir – Quality Assurance
Team Email	sdmay18-37@iastate.edu

Team Website	http://sdmay18-37.sd.ece.iastate.edu/
Version	1.0

Table of Contents

Introduction

Acknowledgement

Throughout development of this project, the team met weekly with Doctor Ahmed Kamal, who proposed the project. He has served as both the advisor and client.

Problem Statement

The problem posed to the team by the parking division was the inability of faculty and staff to locate parking in a timely fashion. This is resulting in time being wasted to search for a parking spot as well as an excess of emissions. The problem occurs primarily during the workday hours 8:00 am – 5:00 pm, these peak hours are when the team is seeking to enact the most change.

To solve this problem the team is proposing to utilize a camera paired with a small computer located outdoors, and a second system to the "heavy lifting". The outdoor setup will use an IP (internet-protocol) camera to acquire images of specific parking spots. The small, outdoor computer will oversee sending the images acquired by the camera to the second remote system and ensure that each camera remains operational by frequent ping checks. The remote system will then check for a car in each spot and update our mobile applications accordingly. Each user will get live updates pushed to their mobile devices to provide a real-time aspect to the service, and to keep each end user appraised of their parking options.

Operating Environment

For each primary component of the system there are different operating environments. Our mobile applications will reside on the user's current mobile device and the domain over which our applications must function is the same that would be expected of any current industry standard mobile device. The pre-processing setup (camera paired with the small computing device) must be able to withstand outdoor conditions, which includes but is not limited to: snow, ice, thunderstorms, humid summers, and high winds. For the camera, we are seeking to acquire an IP66 rated off-the-shelf solution, this is an industry standard assured to be dust-proof and protected from high pressure water jets. The camera must also be battery-powered (similar to Arlo outdoor security cameras) that are able to operate through frigid Iowa winters. The team plans to do continuous research until prototyping begins to ensure the best device is acquired.

Deliverables

Original Solution

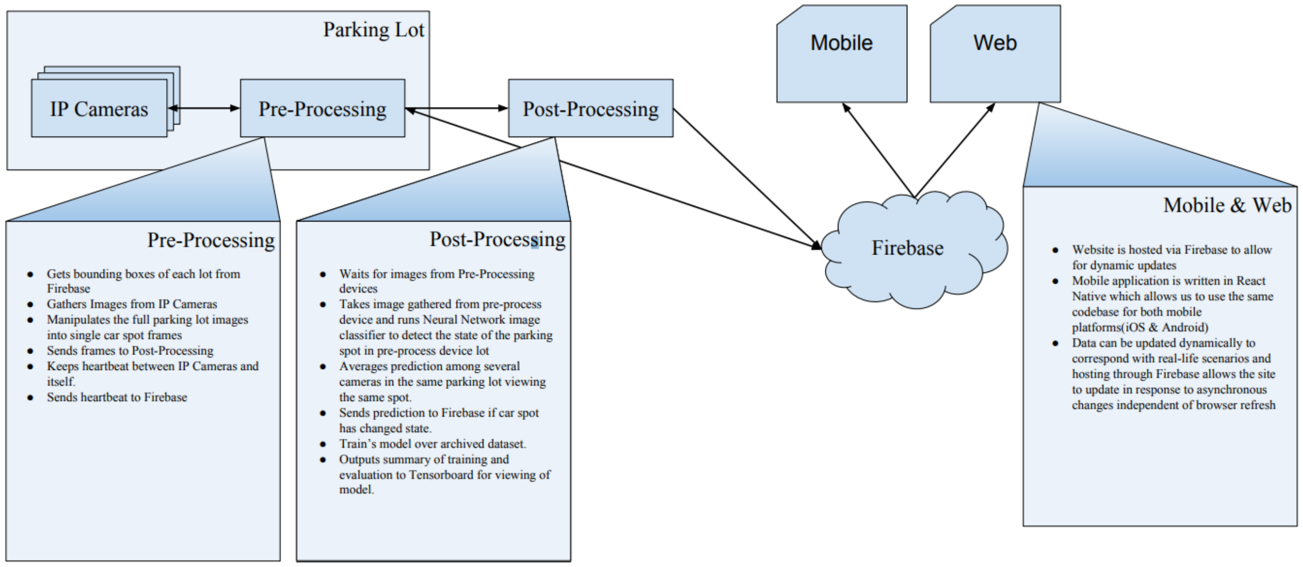
The original project proposal sought to implement a series of sensors in a parking lot that would report the status of specific spots. These sensors would have been installed within the pavement of each spot, incurring both the cost of installing the sensors and the overhead of maintaining them.

Final Solution

The team arrived at an agreement to implement a more modern solution involving a cheap camera and Pre-Processing device paired together, to increase scalability via quick installation in numerous lots and project stability through the ability to have multiple cameras view a single lot to increase prediction confidence. This paired with a Post-Processing machine capable of performing the predictions upon parking lots also allowed the team to implement a project that will keep the future overhead of maintenance low. This solution will be expanded upon throughout the rest of the document.

Design

Design Approach



Pre-Processing

- Gets bounding boxes of each lot from Firebase
- Gathers Images from IP Cameras
- Manipulates the full parking lot images into single car spot frames
- Sends frames to Post-Processing
- Keeps heartbeat between IP Cameras and itself.
- Sends heartbeat to Firebase

Post-Processing

- Waits for images from Pre-Processing devices
- Takes image gathered from pre-process device and runs Neural Network image classifier to detect the state of the parking spot in pre-process device lot
- Averages prediction among several cameras in the same parking lot viewing the same spot.
- Sends prediction to Firebase if car spot has changed state.
- Train's model over archived dataset.
- Outputs summary of training and evaluation to Tensorboard for viewing of model.

Mobile & Web

- Website is hosted via Firebase to allow for dynamic updates
- Mobile application is written in React Native which allows us to use the same codebase for both mobile platforms(iOS & Android)
- Data can be updated dynamically to correspond with real-life scenarios and hosting through Firebase allows the site to update in response to asynchronous changes independent of browser refresh

The design of the final system is still composed of three subsystems. Namely, Pre-Processing, Post-Processing and Application delivery. Each of these subsystems handles a different portion of the workload of the project. Pre-Processing handles the initial acquisition and manipulation of data, before sending a collection of images to the Post-Processing machine. This next machine performs all of the "heavy lifting" of the system, and performs the scanning of images to determine whether a car is parked in a particular spot. Lastly the Application delivery handles the portrayal of real-time data to all users through both Android and iOS mobile applications, as well as a web application.

Implementation Details

Post Processing

Training, Evaluation and Prediction

The post processing machine mainly runs a Python 3 CLI that has several main functions including but not limited to training the model on the dataset, evaluation of the current model, and starting / stopping of current predictions. This CLI was developed as a way to easily interact with our detector and to manage its functions. The training of the neural network includes a batch size and training epoch as parameters to the function. The evaluation takes the images from the evaluation data set and runs an accuracy and AUC on the data set. The prediction of the post processing consists of looking in a directory for camera directories and setting up a network of cameras and pre process device listeners that waits for input from the pre processing devices.

Tools

There are also a couple of tools we implemented for ease of use of our system. This includes a data set creator which makes the data set out of full parking lot images and then applies the bounding boxes to the images and splits them into single parking lot images. Another tool is the camera viewer which allows us to view the cameras with the current spots to manually verify if a car has entered a spot or not. The last tool is a way for the developers to create bounding boxes of parking lots in an easy fashion based upon the camera angle and location. This is done through processing an input image from a camera and drawing bounding boxes around the parking spots.

Pre Processing

Image Capture and Error Reporting

The basic goal of the pre processing device is to capture and split the images from cameras placed in the lot, and to deliver them to the post processing machine. Once a image is captured, images are then created for individual spots and formatted in a way that enables better detection by our model. This is done using OpenCV and Python. Email alerts are live so that when a pre processing device goes offline, there will be some communication as to the device status before the interruption.

Tools

To enable rapid deployment of new devices, we have made interactive scripts that should enable anyone to set up a new pre processing device in a short amount of time. To help test a certain pre processing device we have a CLI that can be used to debug the program and step through all different parts of the system. When a new camera is added to a lot, we have a GUI that enables a user to quickly create the bounding boxes that are included in the cameras view.

Mobile and Web

Backend

The backbone of the mobile and web platform is the reactive database system, Firebase. This system provides our front end live updates as our post processing machine changes the decision of whether there are cars in the spot or not. This is so that the user doesn't have to wait or reload the page / app when there is new data and thus can get live feeds from wherever they are located. Firebase also allows for advanced analytics so we can track user interaction and locations to learn more about how our applications are being used. With this information from Firebase we can fine tune our product and develop better features for our users.

UI

The Mobile platform is written in React-Native such that it is supported on both Android and iOS platforms. The Web platform is written in Angular. Both of these frameworks allow for the reactive nature of our application to be easily implemented. The UI is centered around locations that currently support our system such that the user can access the current school they are at and the parking lots that are supported and view what the current status of the parking spots are. The maintenance of the application is also displayed to the user when the lot or school is under maintenance. Both the mobile and Web platforms show an overhead view of the parking lot grid for easy spot finding while looking for a spot.

Testing

All of the code developed over the duration of this project has went through a series of merge requests. These were utilized as a safeguard against poorly written code from entering out project. Each contribution must be approved by at least two other people on the team. These approvers usually consisted of people working on the same aspect of the project, or a lead (if these failed then someone with past experience). The role of the approver was to look over the code to ensure that it met the groups coding standards, as well as run test/edge cases against the code to ensure it performs as intended. Having implemented this process the group has established thorough coverage of the codebase, and has limited the number of bug fixes needed throughout the development lifetime of this project.

Pre-Processing

When developing for the pre processing device, we used extra raspberry pis as test environments where changes could be tested before they were pushed to the production pi. This let us keep the service running correctly without having to stop it to test, as well as possibly introducing false data into the system in the event of any unfixed bugs.

Post-Processing

When testing the post-processing side of our project, a lot of it was ensuring that we met our accuracy mark of 90%. This meant having a dataset comprised of quality images from our test lot, spots with and without vehicles in all climates to ensure that our model can make accurate predictions not solely in ideal conditions. In order to do this we manually went through the archived full images we take every hour, and selecting images that give us a good variety of different situations (snow, night, rain, double parked, etc). We comprise these split images into an overall dataset and separate into a training and evaluations set. We train our model on the training set, and to ensure that we are meeting the 90% accuracy mark we run our model and make predictions on the evaluation set. After testing on our evaluation set we got an accuracy of 93%. Aside from that we also do manual testing by running our CameraViewer tool, which gives us a live view of the lot and overlays the predictions on the appropriate spot. This allows us to constantly check on the state of the lot and if the model is making accurate predictions.

Mobile

Early on in development of the mobile application we had issues with getting the development environment set up to work on both Android and iOS using the React Native framework and the Firebase platform. Given that both React Native and Firebase are very new technologies that are still getting frequent updates by their developers, it was inevitable that we would run into compatibility issues when adding new modules for our app to use. To ensure that both Android and iOS apps ran and operated as the same, both of the developers working on the mobile app would pull the changes made to the mobile app and run it on their respective device. If there were issues getting the app to run on iOS or Android, then a fix would be applied, followed by testing the app on iOS if there was an issue on Android, or vice versa.

Web

To test the website using Angular it was pretty simple to view changes iteratively through the ability of “ng serve”. This locally hosted the website on a developer’s machine and allowed functionality to be tested as it was built and then updated. Issues that occurred due to interfacing with Firebase from the website were also able to be solved in this manner, as references to data and the “reactiveness” of the website would be able to be tested as an end-user immediately after it was developed. Similarly as new functionality was implemented in the website testing in this manner allowed for the ability to test code without having to host it as a live URL which helped the team avoid hosting any buggy or poorly performing code.

Outcomes (Status)

Subsystems

The subsystems of the project have been implemented to uphold a satisfactory level of performance. These have allowed the team to establish a concrete "proof of concept" to demonstrate the success of the overall system design as well as to establish the benefit that this project could bring to potential customers and users.

Status

The final status of the developed system dubbed "LiveSpot" was developed to an operational state of completion with all core functionality implemented. The only issues were the acquisition of a suitable outdoor camera solution in conjunction with the ability to mount this camera to a light fixture. However four group members have met with Randy Larabee (FPM) and have planned a meeting with Mark Miller (Parking Division) and Randy to discuss a remedy. These four members have planned to continue developing this project to implement at Iowa State under the assumption that all personnel involved are interested in working with us.

Existing Projects and Literature

Throughout the development of this project the team has remained cognizant of two similar pre-existing projects. A similar concept had been carried out by Andrew Sobral titled "Automatic Parking Lot Classification" [1] and the link has been provided below for convenience. The team noticed however that this solution was only trained for sunny, cloudy and rainy days and does not seem to handle snow, and seemed to run with a lot less hardware and processing involved. Fundamentally however was that this concept lacked the ability to dynamically (and asynchronously) appraise users of data driven changes on the front-end. A second project[2] was discovered and was more closely related to what the team tried to achieve. However we developed a more sophisticated parking lot description that allows users to quickly and easily distinguish which spots are vacant without the necessity of memorizing spot identification numbers. This second project was more closely aligned with the teams preliminary architectural design decisions, and unlike the aforementioned project, it did also incorporate a mobile and web interface.

Appendices

Appendix I

Post-Processing

Command Line Interface

The post processing machine main purpose is to provide the developer with a command line interface that controls the interaction of the neural network model. This is done through a set of different commands listed below:

- help : help page
- data collect_img_net : Collect Image Net Data
- data clear : clear all data
- data init : Data initialization
- model train <epoch> : Trains model based on epoch and batch size
- model eval : Evaluates model for accuracy and loss function
- model predict_start : Runs prediction on predict_imgs in data directory
- model predict_stop : Stops prediction on predict_imgs in data directory
- model clear : Clear model
- config : Load config
- logs clear : Clear all logs
- q : to quit

Tensorflow

For our neural network model development, we use a piece of open source software called Tensorflow by Google. This allows us to train, evaluate, and predict neural network models while also specifying the right parameters to give us the highest accuracy and best AUC. It also allows us to use the GPUs on the system to run predictions and train which allows the processing of images to be in real time. To install Tensorflow follow the installation tutorial located online or in the wiki page linked with this report.

Tools

See Bounding Box Creator for more details on how the creator is used.

For the Camera Viewer, this tool needs a connection to the post processing device and takes in two parameters into the command line of running the script *CameraViewer.py <lot_id> <camera_id>*. This allows the developer to see the live view of the parking lot along with the current predictions of each spot to see if the accuracy holds on the live feed. Pressing 'r' while in the tool refreshes the page.

Pre-Processing

Setting up Pre-Processing Device

Live Spot pre-processing is designed to run on a raspberry pi running the Raspbian OS. We recommend at least a Model 2, as we have tested processing on the older models and it does not perform near as well. It would be possible to run it on another type of small Linux based machine, to do this you would need to edit the Serial.py file to grab the serial number accordingly.

You will need to have Open CV compile and ready for use with python 3 on the device. There are many guides on-line for this process and it is recommended that you do this by hand yourself, but we have provided a script that can do it for you. It will install OpenCV v3.1 for python 3. Installing with pip install OpenCV-python will not work and will break any existing OpenCV installation.

We have created an easy to use install script and it can be ran with ./InstallLiveSpot <Install Directory>. You will need to have your post processing machine set up before you run the install script as it will guide you through setting up a secure connection between the two. You will also need to have your firebase credentials JSON file on the pi and have the location ready as you will be configuring the credential file during installation.

At the moment you will need ISU credentials to pull the LiveSpot code.

Currently the pi will use any webcam connected to it via USB for the video input. Whatever quality of webcam you use the image will be taken in 720p.

After installation, you can restart the machine and the service should start up automatically after a successful boot.

Setting up Bounding Boxes

To set up the bounding boxes for a particular camera view, you will first need to gather a test image from your device. To get one in the same format that the Live Spot service will use, you can run the command fswebcam --no-banner --set brightness=65% -r 1280x720 <Output File>. You can then move the image to a local machine to run a tool we have created for easily setting bounding box coordinates.

How It Works

The pre-processing service runs in the background on the pi. Using a webcam that is connected, it takes a picture of the lot. Based upon bounding boxes that are set in the Firestore database, images are clipped up into individual spots. They are grey-scaled and made into 256x256 images. After all spot images have been created for one round of image captures, the images are transferred to the post processing machine. If the post processing machine has not digested the last round of images, no new images will be transferred. Once the transfer has completed or been skipped, the process starts back over again.

Once every hour there is a heartbeat check where the pi makes a connection to the database to update the status of the pi as online.

Android Application

These are the steps Joe has used to run the mobile app on Android (minimum Android API level is 16) using Windows. Make sure you have NodeJS, npm, Python2, and Android Studio installed.

Click this link and click on "Building Projects with Native Code" to make sure you have all the React Native dependencies installed correctly.

You can also run the app from a Genymotion Android virtual device. The app has been tested and works on virtual devices with API levels 23 and 24. Other API levels greater than or equal to 16 should work, just haven't tested them. For some reason virtual devices with API level 26 couldn’t run the app.

- cd sdmay18-37/ReactNative/LiveSpotApp
- run npm install. All the necessary node modules will be installed.
- Either connect a phone to computer or start an Android virtual device using Genymotion.
- run react-native run-android. The app will begin building and downloading dependencies.
- This is the React Native packager server that bundles the JavaScript files together and sends it to the app to run.
- At this point, the app should be installed on your device. After opening the app and waiting a few seconds, you should see a red screen.
- If you are using Genymotion, then the first thing that pops up should be a screen asking to permit the app to draw over other apps. Enable this option then select the app from the app drawer.

- This is the in-app developer options menu. This menu allows you to reload the JavaScript files, enable live reloading, run in debug mode, and access developer settings. Press CTRL+M to access this menu using a virtual device. If you are using a physical device, then shake it. To see your changes made instantly, enable both "Live Reload" and "Hot Reloading".
- If you are using a physical device, then you will have to connect your device to the React Native packager server (make sure both your computer and device are on the same network). To do this, go into the in-app developer options menu and select "Dev Settings".
- Now select "Debug server host & port for device". Now type in "X.X.X.X:8081" where X.X.X.X is the local IP address of your computer that is running the React Native packager server. Select "Reload" from the in-app developer options menu then the app show now load onto the device.
- It is likely some steps are missing since there are lot of things that could go wrong and it is hard to keep track of which little changes work. Feel free to add/edit any steps that you took to get the app running in your environment. If you have any trouble or questions about installation then ask Joe and I'll try my best to help out

Bounding Box Creator

This tool allows you to edit/create new bounding boxes for parking lots and automatically export them to our database.

When running the tool, it will initially ask for you to enter a lot and camera. It will then ask you to select an image of the lot.

- If you want to edit the boxes for a current camera, then enter one that is in the database, and they will automatically overlay the picture.
- If you want to create boxes for a new camera, then enter one not in the DB and the image will show up blank.

Two tools will pop up once your image is selected, it is your CanvasViewer, and your Updater. The CanvasViewer, shows the image, the bounding boxes, and it is what you click on to add new points. The Updater has buttons so you can add a new box, delete current boxes, edit a box, delete all boxes, and send the coordinates to the database.

Functionality

- To create a new box, enter a digit into the small box in the updater and click "Add New Box". You can now left click on the CanvasViewer to add data points, represented as gray circles. If you want to move the data points click on them and drag with your cursor. (This will automatically update the coordinates) Once you have your 4 data points in the desired area, hit "Enter" on your keyboard. Your new box will be shown on the CanvasViewer and will be added to the list on the Updater.

- To edit, select a spot id from the list in the Updater, it will highlight in blue, and click "Edit Coords". Move the circles in the CanvasViewer, and hit "Enter" on your keyboard when done.
- To delete a spot id, select one from the list in the Updater and click "Delete Spot ID". (Spots cannot be deleted if being edited)
- To delete all spots, click "Delete All Spots". A prompt will ask if you are sure, click "Yes" and all spots will be deleted.
- Click "Export to DB" to send bounding boxes to database.
- If you double click a spot id in the list, it will print its coordinates in the space above the buttons in the **Updater**.

What will prompt an Error Message:

- Not clicking "Add New Box" before trying to click on CanvasViewer.
- Hitting 'Enter' without laying down 4 data points for the bounding box.
- Trying to delete a spot that is being edited.

There is more documentation covering all parts of the system in more detail at https://git.ece.iastate.edu/sd/sdmay18-37/wikis/home

Appendix III

Development Considerations

The hardest part about this project was the obtainment of hardware to use for setting up cameras in the lots. When planning how the cameras should be set up, it was hard to find a solution that was a quick semester implementation. The client's original proposal dealt with using sensors in each spot was going to take a lot of overhead too so we thought reducing the overhead costs of putting sensors in each spot was the way to go. This brought on other struggles such as where to put the camera and how to mount it. Initial suggestions were mounting it to a tree, mounting to light pole, or mounting to building. The best, out of these options, was the light pole due to it having power already inside the pole.

Future Considerations

The next steps of the process will be refining pieces of our design in such a way that makes our system efficient and accurate. These steps include improving the overall design of our mobile and web application to use overlays on the google maps api, gathering more data and improving accuracy of predictions, modifying model to adjust to our problem need's, and implementing object detection api as another way to detect the car's features.

In terms of the hardware features, implementation into a parking lots is of high importance. As such the plan includes keeping up to date with our client the progress of the camera systems that are placed and continually improving our design to meet the camera system.

Closure Materials

References

[1]A. Sobral, "Behance", *Behance.net*, 2017. [Online]. Available: https://www.behance.net/gallery/29828109/Deep-Learning-Automatic-Parking-Lot-Classification. [Accessed: 03- Dec- 2017].

[2]S. Valipour, M. Siam, E. Stroulia and M. Jagersand, "Parking Stall Vacancy Indicator System Based on Deep Convolutional Neural Networks", *Arxiv.org*, 2016. [Online]. Available: https://arxiv.org/abs/1606.09367. [Accessed: 03- Dec- 2017].