Ben Fernandez, Kyle Rood, Suraj Shah

5/5/2020

AmberVision Final Report

## Section 1: Team member names, and a description (a list) of who-did-what in the project.

* Ben Fernandez: Ben was responsible for the frontend application
  + Created frontend application in VueJS
  + Bundled together different widgets to create main components: Map Page and Query Page to view AmberVision detections
    - Map Page:
      * Used Bootstrap CSS framework for styling and grid-based layout
      * Map was created through Leaflet framework and is marker-based interactive map of Washington DC
      * When a marker was selected, users can view the camera image, detections in the scene, and camera metadata
    - Query Page:
      * Queries based on size and color were used to filter AmberVision detection dataset and show users historical camera images
* Kyle Rood: Kyle was responsible for the machine learning algorithm
  + Car Detection / Size Identification using Yolov3
    - Yolov3 is the fastest detection algorithm available and allows us to have accurate detections as fast as possible.
  + Color identification using PCA Embedding + K-Nearest neighbors
    - The PCA embedding took cropped images of cars as a 200 x 200 vector and embeds them into a lower-dimensional space to highlight important features. We embedded thousands of test images together to give us a model for car colors. Our system embeds a new image using that model and assesses the color using K-Nearest Neighbors. We compute the distance between every single point and pick the three closest points to each point. The color of these closest points that appears most often is the predicted color of any given car.
* Suraj Shah: Suraj was responsible for the middleware connection between the machine learning algorithms and the front-end program.
  + Created image collection python program that uses the TrafficLand API and stores images
  + Created API for middleware connection between front-end and backend
  + Managed MongoDB and allows communication between machine learning detections and the database. YOLO gets detections of the vehicles and then uses API to POST data to the database that front-end can use
  + Frontend API to display the data needed for users to see the detected images, their respective detections including color, size, etc, and previous history of all other detected images

## Section 2: A brief non-technical overview of your project (2 paragraphs).

AmberVision is one part of a solution to the question: how can we leverage existing technology to search for Amber Alert victims? AmberVision has a simple goal -- detect cars and display these detections to law enforcement. With this goal in mind, we aim to use a clear and concise design method so that law enforcement can successfully receive the information we provide. This problem is extremely important because AMBER Alerts were created many years ago, but the lack of technology involved in helping these victims is apparent. With the AMBER Alerts, we see this as a tool that takes in the latest inputs of current AMBER Alerts, and with our tool, law enforcement can track cars that are in the AMBER Alerts. We see this tool being primarily used by law enforcement because of the specificity of the project and the need for an improved version that can help law enforcement scrape through thousands of images using smarter technology that exists currently.

Using state-of-the-art object detection models, we look to help identify those vehicles using a network of traffic cameras and in doing so, contribute to the goal of AMBER Alerts in search and recovery of the child. Our product would function as a web application that law enforcement can use to see realtime information on vehicles on DC roads. We would get this information from a third-party source that provides the data for these traffic cameras. We would also be displaying information about these vehicles such as the size, color, and location of last seen.

## Section 3: The link to your project website, with a list of items on the website.

* Website: <https://gw-cs-sd.github.io/sd-20-fernandez-rood-shah/>
* List of items:
  + DesignHW1.docx
  + DesignHW2.pdf
  + Fernandez-Rood-Shah-HCI-1.pdf
  + Fernandez-Rood-Shah-Writing-2.docx
  + Fernandez-Rood-Shah-Writing-3.docx
  + Fernandez-Writing-1.docx
  + Fernandez-Writing-2.docx
  + Rood-Writing-1.docx
  + Rood-Writing-2.docx
  + Shah-Writing-1.docx
  + Shah-Writing-2.docx
  + DemoOverview30-40-70-90.docx
  + Report and Plan-Fernandez-Rood-Shah.docx
* Final Video: <https://www.youtube.com/watch?v=AO9Qwldba3k>

## Section 4: A complete list of libraries, packages, and APIs that you used for the project.

* Python: Flask, PyMongo, JSON, Cross-origin resource sharing (CORS), urllib (getting images from URLs), hashlib (hashing an image), gevent (multithreading in python)
* Vue.js: Leaflet. Axios, Bootstrap-Vue, Vanilla JS, HTML
* MongoDB
* Sklearn, cv2, Matplotlib, Pandas, Scipy

## Section 5: A brief *technical* overview of your project (2 paragraphs).

The machine-learning algorithm was separated into two parts. The first part was Car Detection and Size Identification using the Yolov3 framework. This is a real-time detection algorithm and helps our system to be as fast as possible. The second piece is a PCA embedding, which is used to classify the color of the car. The PCA embedding took cropped images of cars as a 200 x 200 vector and embeds them into a lower-dimensional space to highlight important features. We embedded thousands of test images together to give us a model for car colors. Our system embeds a new image using that model and assesses the color using K-Nearest Neighbors. We compute the distance between every single point and pick the three closest points to each point. The color of these closest points that appears most often is the predicted color of any given car.

Once the machine learning algorithms were used on the detected vehicles, there were two important aspects that need to be discussed both before and after these detections. The first was the data collection that allowed machine learning to detect the data. Using our database image collector Python program, we used HTTP requests on the TrafficLand API and retrieved the metadata and image of the traffic cameras. We stored this in a JSON format in a MongoDB which allowed for easier queries due to the flexible nature of NoSQL databases. Once we had this data, we created an API Python program built with the Flask web development server to GET and POST data back and forth between the NoSQL database and our frontend and machine learning. The machine learning submitted the data via a GET request, compiled its results, and submitted a POST request to submit the detection points for each specific image.

On the frontend side, the user would be presented with a map of Washington DC upon landing. The user would be able to select a marker of a camera on the map and be presented with the latest image from the AmberVision detections all funneled by the API created. The data received from the GET request would be processed and formatted for a seamless user experience when viewing camera images. On the same map page, the user could all see the historical images from that camera as well as camera metadata stored such as location, latitude/longitude, and the last time the camera was updated. In addition to the map page, the user could search through the historical data collected by AmberVision based on filters such as size and color. This frontend application presented the users with the most important information first and allowed for a high level of interaction to help display data that could be useful for Amber Alerts.

## Section 6: One "if I had to do this again" paragraph from each team member that contains *technical* lessons learned, and what would you have done differently.

Kyle

If we had to do this again, we would definitely have experimented more with our color detection. The car detection and size detection worked well enough, but our color detection only ended up with an accuracy of 24%. One way to improve this would have been to have more labeled data. We had around 2000 labeled images that contributed to our embedding, so it may have been helpful to increase the number of images in our model. A way to improve the algorithm itself would have been to visualize exactly what the embedding looked like, in order to debug our embedding of images. Our initial tests showed a successful segmentation of images based on color, but we could have added a way to visualize the original pictures associated with the data points. Though this was in some ways a failed experiment in some ways, this algorithm was very fast, which helped to achieve one of our goals. Our color detection algorithm definitely has potential and should be pursued in the future.

Suraj

If we had to do this again, I definitely would have looked into hosting this in the cloud instead of on local machines. We began our development process on a local machine with a GPU because that was the only machine capable of the calculations needed to handle the machine learning algorithms. However, we didn’t look too much into cloud options because we were under the assumption that the cost would be a significant barrier to accessing these options. I believe if we had used AWS and Google Cloud Platform to perform our calculations and host our web application online, we would have been able to avoid the implications of COVID-19 entirely because our platform would have been remotely accessible. With our current machine, we were limited to SSH, and the lack of visual communication between our machine and the results limited our progression to develop the web app. Furthermore, we couldn’t run the programs locally on our own personal computers because we did not have the resources necessary to accurately detect the vehicles — the CPU’s of our laptops were not powerful enough to quickly compute the thousands of detections at once. Therefore, with hindsight being 20/20, I would say the biggest technical lesson learned is using the cloud for intense computations that cannot be done locally to be able to access the information remotely.

Ben

If I had to start over and/or continue the project, I would spend more time focusing on the analytics for the frontend users. The level of data being collected from AmberVision was of high fidelity and a lot could be parsed and determined from the detections. For example, at any given time in Washington DC, we had the exact locations of cars in the municipality as well as their direction and density. This information could be helpful to track not only the cars in the scene but also the movement and routes of vehicles in the scene. On top of this, I would have displayed more tabular and textual data in addition to the sole images soon. The current project has a heavy focus on just the display of the images, but there is space for additional tags and information to more easily present the data from the detections. Overall, the framework and technologies I would use would stay consistent and try to go deeper with more data presented to the user!

## Section 7: Instructions for follow-on projects. Write a detailed set of instructions for the next group of SD students who choose to build on your project:

* + How to download and get the project working. If there's equipment, a description of where to purchase (what model #, etc).
  + What works, what doesn’t, what to be aware of (pitfalls, issues).
  + Ideas for next steps

To run this project, simply look in the ReadME.md of our GitHub page. There are a few steps to set up the environment, and a couple of commands to run the two pieces of our application. As long as you have a computer with decent processing power, you should be fine! Some of the machine learning libraries require a GPU, so there may be some problems running this on a machine with only CPU capabilities. This was one of the biggest issues we ran into, so using a machine with GPU capabilities is highly encouraged. No equipment is necessary.

This application is definitely close to what we wanted to accomplish from the beginning. There is a pipeline that connects the data collection, machine learning, database, and front end aspects of the project. We can have all aspects running together, and have a processing time (from data collection to publishing results in the front-end) of around one minute.

The front end has all of the information we wanted to display, but this could potentially have had more analytics for the user. We had one idea about a heat map displaying traffic in different areas by counting the number of cars we detected. This could potentially give a more accurate view than Google Maps because we would be counting every car, not just the ones that have Google Maps.

The color detection algorithm can definitely be improved. Like we said above, more data and better visualization of our results would have helped to understand what exactly was going wrong. We attempted to use deep learning technologies but did not find success in speed or accuracy. There may be other technologies that have come out since then which could help one of these problems, making deep learning viable for this project.