Christopher Highman

Subject: You picked the WRONG engineer - a call to arms - anyone interested in joining forces ?

From: Christopher Highman
Sent: Tuesday, April 7, 2020 3:23 PM
To:
Subject: You picked the WRONG engineer - a call to arms - anyone interested in joining forces ?

On March 20th at 5:49am, my car was stolen outright from my driveway just minutes from the office. Police stopped by long enough to take a report and leave. We were told they will be actively looking for our car. Two weeks goes by and we leave trust in the system. I did not have comprehensive coverage for this vehicle so there is no insurance claim possible.

← Crime & Safety

C.C. Highman
Lookout Rdge and Perrigo Heights · 20 Mar

Our car was stolen, please check your cameras!!!

Around 3:45am this morning, someone burglarized our Red Outlander and stole our Light Blue Toyota Prius C. There was a broken keyfob that could start the car inside. Our dogs were going nuts around that time in the garage and we let them inside. This person was brazen enough to take their time with dogs making tons of noise.

We have made a police report and ask that any neighbors who live near us please check your cameras and see if you notice anyone suspicious or see our light blue prius.

Any help is greatly appreciated!
. I have reported this information to the police.

Description of vehicle involved – Color: Light Blue, Make: Toyota, Model: Prius C, Year: 2012, Type: Car, License plate: Indiana STH577, Other details: Last seen in driveway at 9545 173rd PL NE early morning of March 20th. We are corner house from 95th across from Armory. Suspected time of theft is 3:45am.

Thank Comment 2 9

C.C. Highman, Lookout Rdge and Perrigo ...

Here is a picture of the stolen vehicle

20 Mar Thank Reply

We initially thought it was stolen around 3am but a neighbor in the area forwarded this image that confirmed time of theft was 5:49am

Take a look.

Your Bonus Room camera noticed some activity at 5:49 AM on 3/20/20.

CHECK LIVE STREAM

Last Thursday, we checked out toll bill and noticed a charge from 3/20 at 6:02am which provided another clue as to what happened to our car.

I become curious and start checking sites like NICB which allows VIN lookups to see if they report as "stolen". Our VIN still showed up as not stolen. I called GoodToGo and they confirmed they have a picture of our vehicle crossing at that time but it only shows the back of the vehicle. They claim they likely have video footage but I would have to go through WDOT to see if its possible to obtain. I called back the Redmond police and asked for an update. Nothing. Knowing now there has to be cameras along the way and based on 14 minutes distance from my home to the toll, they had to take a direct route to get there. I called the Washington State Police and other places asking about the ability to use these cameras to locate my car. They said it wouldn't help find it! I know this is not true after my own investigation. Seattle PD uses "ALPR" technology that captures license plates in 360 degree and automatically compares them against a database of plates of interest.

Seattle Policy on ALPR tech.

www.seattle.gov/police-manual/title-16---patrol-operations/16170---automatic-license-plate-readers

Seattle.gov
Mayor Jenny A. Durkan

16.250 - Interaction with the University of Washington Police Department

16.300 - Patrol Canines

3. Authorized and Prohibited Uses

ALPR systems will only be deployed for official law enforcement purposes. These deployments are limited to:

- Locating stolen vehicles;

- Locating stolen license plates;

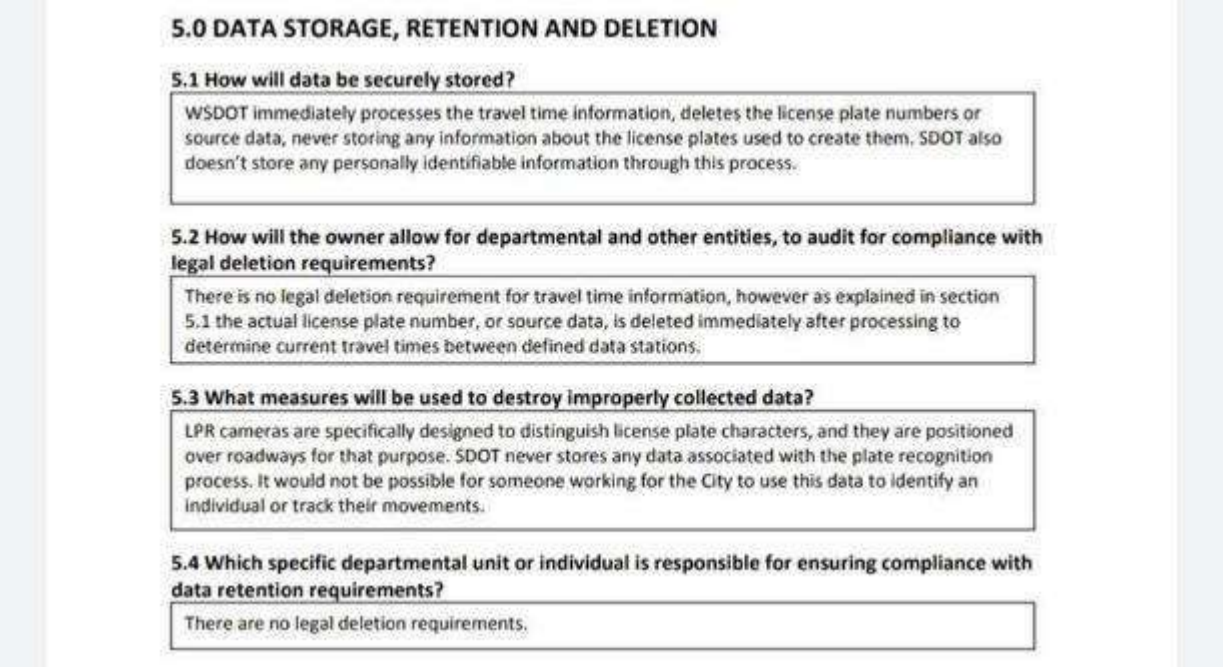
- Locating wanted, endangered or missing persons; or those violating protection orders;

- Canvassing the area around a crime scene;

- Locating vehicles under SCOFFLAW; and

- Electronically chalking vehicles for parking enforcement purposes.

So, what about all the other cameras out there? SDOT makes them **available for the free use of the public.**



5.0 DATA STORAGE, RETENTION AND DELETION

5.1 How will data be securely stored?

WSDOT immediately processes the travel time information, deletes the license plate numbers or source data, never storing any information about the license plates used to create them. SDOT also doesn't store any personally identifiable information through this process.

5.2 How will the owner allow for departmental and other entities, to audit for compliance with legal deletion requirements?

There is no legal deletion requirement for travel time information, however as explained in section 5.1 the actual license plate number, or source data, is deleted immediately after processing to determine current travel times between defined data stations.

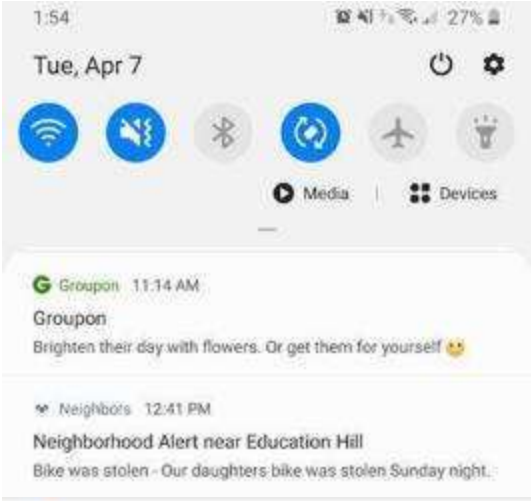
5.3 What measures will be used to destroy improperly collected data?

LPR cameras are specifically designed to distinguish license plate characters, and they are positioned over roadways for that purpose. SDOT never stores any data associated with the plate recognition process. It would not be possible for someone working for the City to use this data to identify an individual or track their movements.

5.4 Which specific departmental unit or individual is responsible for ensuring compliance with data retention requirements?

There are no legal deletion requirements.

I began talking to several people who claim Redmond area was seen as a great target for car thieves from Seattle. I began doing anecdotal searches and looking at statistics. The Seattle area is often in the **top 10** for property crimes.


1:54
27%
Tue, Apr 7
Media
Devices
Groupon 11:14 AM
Groupon
Brighten their day with flowers. Or get them for yourself
Neighbors 12:41 PM
Neighborhood Alert near Education Hill
Bike was stolen - Our daughters bike was stolen Sunday night.


Max Edwards, Grasslawn
Suspicious activity at 435 AM
Strange behavior in our front yard. Description of person involved – Hair: Too dark to tell
4 replies · 3 days ago · 48 neighborhoods
Thank
Comment
3
4


Kerry Lebel, Grasslawn
Package stolen @ Marymoor Trails
first our car gets broken into and now expensive package gets sotelo within minutes of being dropped off. :(http... See more
Package stolen in Marymoor Trails
neighbors.ring.com
11 replies · 3 days ago · 47 neighborhoods
Thank
Comment
3
11


Coco Davis, Willows · Rose Hill
stolen mail by LWTC
I went on a walk this morning and saw a pile of mail on 132nd at Lake WA Tech College. Some seemed like it wa... See more
+2
25 replies · 6 days ago · 50 neighborhoods
Thank
Comment
21
25

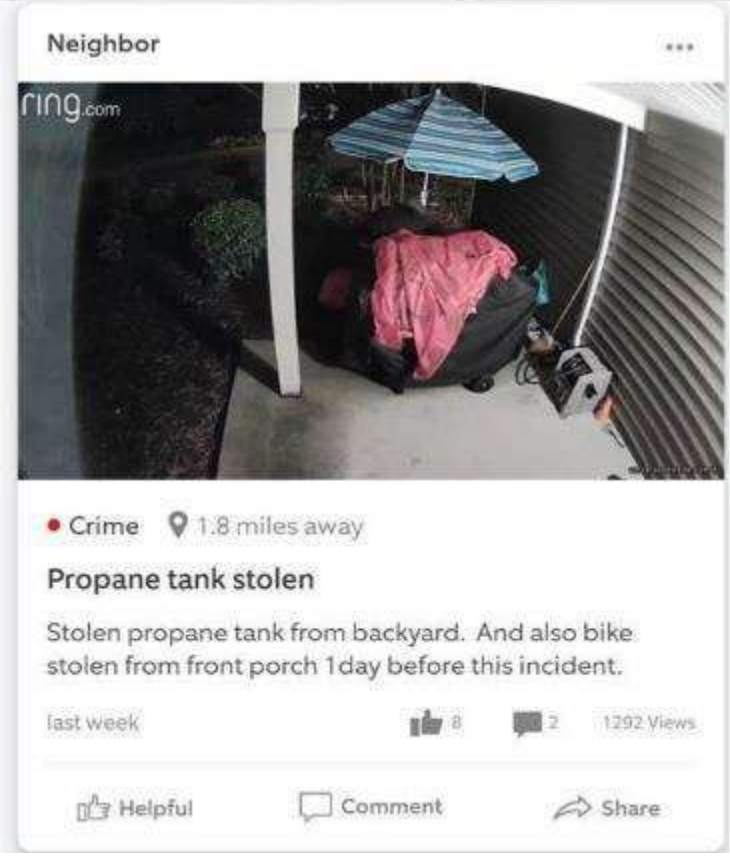
KUnal Tare, Grasslawn

Car number plates swapped with stolen ones

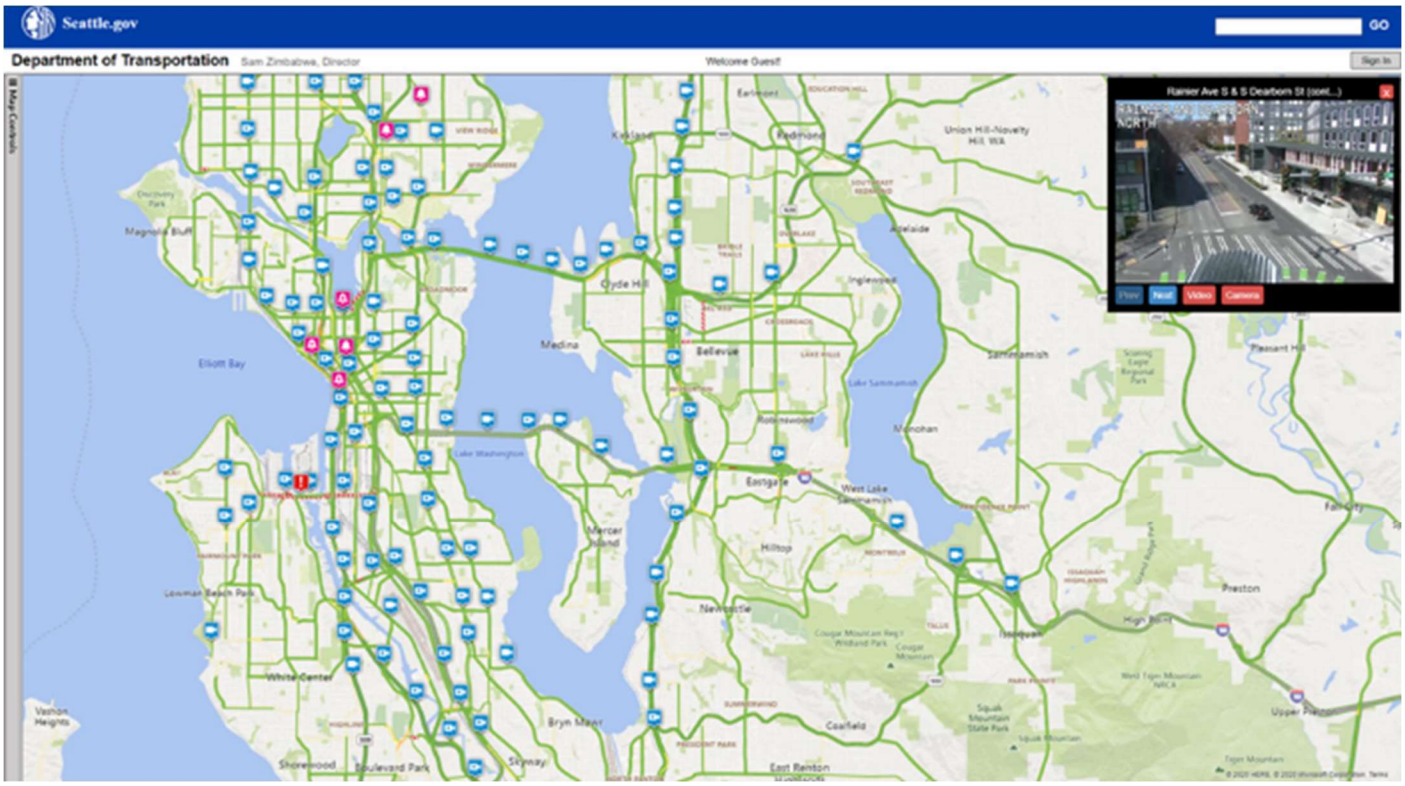
Someone stole plates from my car and replaced them with some other number. I reported to police and learnt... See more

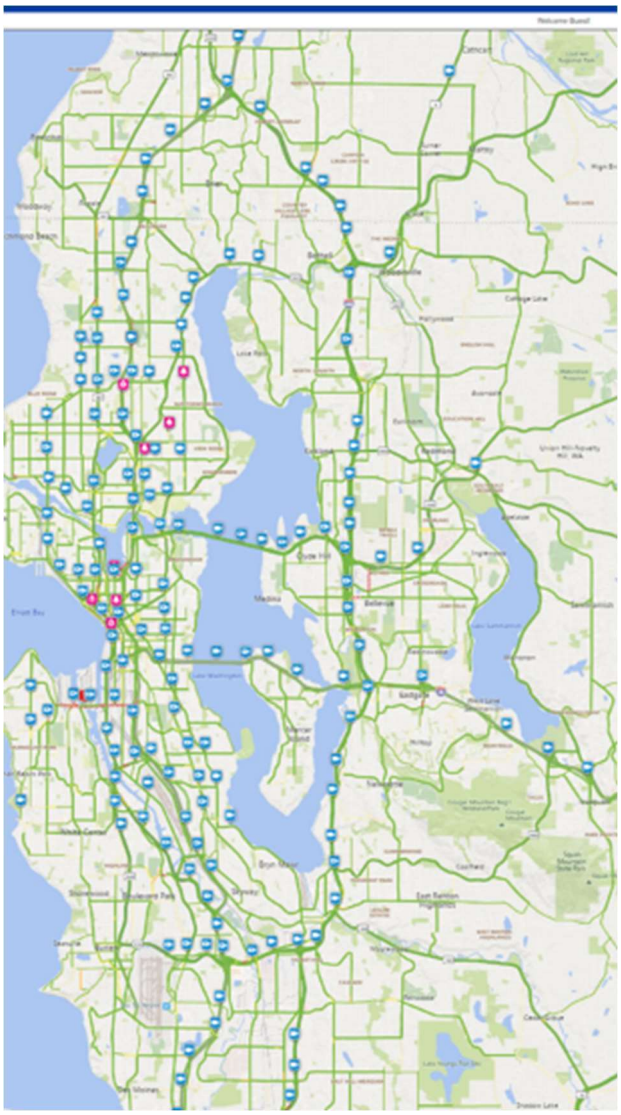
16 replies · 25 Mar · 48 neighborhoods

Thank Comment 8 16



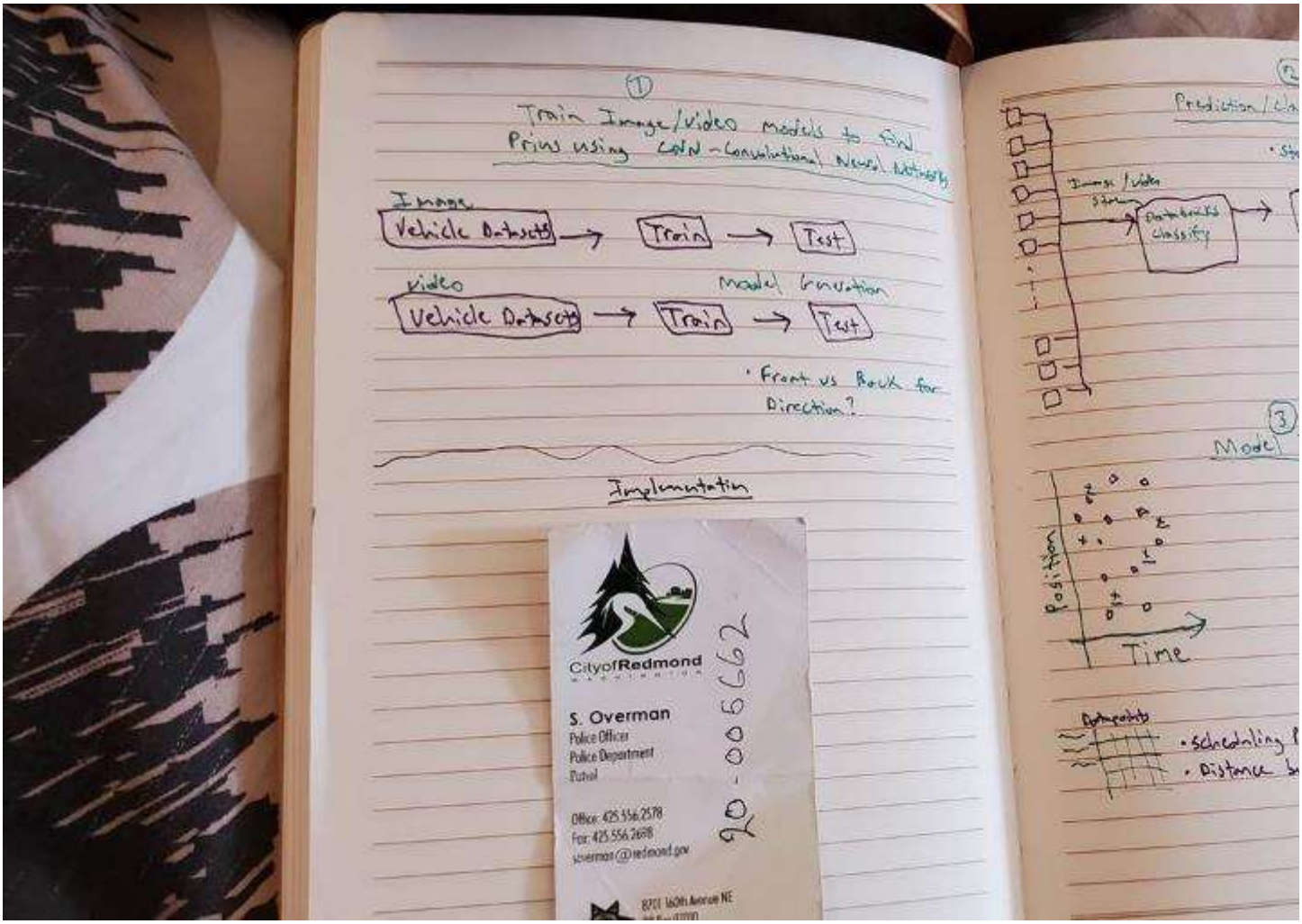
It became apparent people were getting taken advantage of in these areas because of little ability for law enforcement to do anything about it. I started wondering just how many cameras are out there. **What I discovered left me breathless.**


Seattle.gov
GO
Department of Transportation
Sam Zimbabwe, Director
Welcome Guest!
Sign In
Rainier Ave S & S Dearborn St (cont...)
Prev
Next
Video
Camera
Elliott Bay
Magnolia Bluff
Queen Anne
Kirkland
Redmond
Union Hill-Novelty Hill, WA
Clyde Hill
Medina
Bellevue
Issaquah
Sammamish
Pleasant Hill
Monohan
Eastgate
West Lake Sammamish
Mercer Island
Hilltop
Newcastle
Preston
Cougar Mountain Regional Wildland Park
Cougar Mountain
Squak Mountain State Park
Coalfield
Tiger Mountain
Upper Preston
Lake Washington
Lake Sammamish
Lincoln Park
Lemman Beach Park
White Center
Vashon Heights
Bryn Mawr
Skyway
Shorewood
Boulevard Park
East Renton



There were **hundreds** of these publically accessible cameras in the Puget Sound area. Many of them had high resolution, live video feeds. **There were a total of 1,961 cameras in the Puget Sound area available to the public. Last Friday evening, I decided I was going to find my car.**

As any seasoned engineer would, I carefully outlined my plan to capture frames from every camera available on a regular cadence and run them through pre-trained **convolutional neural networks** designed to identify vehicle make, model, and color with **stunning accuracy.** There simply are not that many 2012 light blue Toyota Priuses floating around. This would allow a way to plot vehicle movement. My car in particular has several noticeable features on the body that would distinguish it apart.



After just a few hours, I had already created a dataset having every camera by GPS location, description, and camera URL. The same JPG was refreshed every so many seconds. I learned if the type was "SDOT", it meant a live HLS (ts) feed was available. If it was "WDOT", only camera images were available.

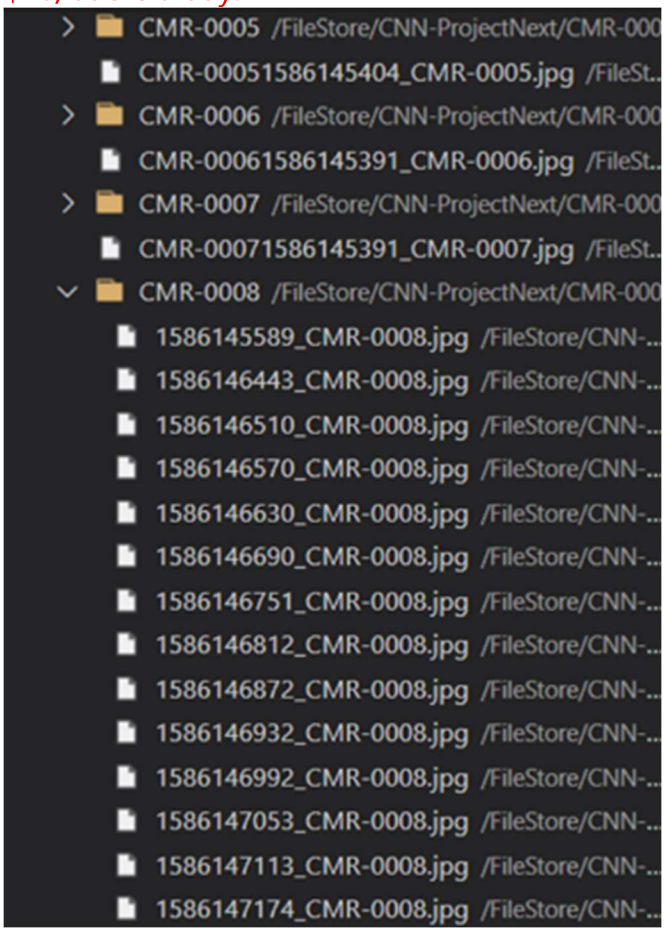
c: > Users > chhighma > Downloads > result.csv

```
"PointCoordinate__001","PointCoordinate__002","Cameras__Id","Cameras__Description","Ca
"47.526783365445","-122.392755787503","CMR-0112","Fauntleroy Way SW & SW Cloverdale St
"47.5611064685211","-122.38677811046","CMR-0038","42nd Ave SW & SW Alaska St","42_SW_A
"","","CMR-0093","California Ave SW & SW Alaska St","California_SW_Alaska_NS.jpg","sdo
"47.5756262519552","-122.386760647909","CMR-0252","California Ave SW & SW Hanford St"
"47.5811953428481","-122.386545754984","CMR-0250","41st Ave SW & SW Admiral Way ","41_
"","","CMR-0251","California Ave SW & SW Admiral Way","California_SW_Admiral_NS.jpg","
"47.5611250678466","-122.38148149011","CMR-0113","Fauntleroy Way SW & SW Alaska St","F
"47.6906091088433","-122.376806120068","CMR-0008","15th Ave NW & NW 85th St","15_NW_85
"47.6763573642042","-122.376758804153","CMR-0007","15th Ave NW & NW 65th St NS","15_NW
"","","CMR-0006","15th Ave NW & NW 65th St EW","15_NW_65_EW.jpg","sdot"
"47.6485862077929","-122.376378124213","CMR-0012","15th Ave W & W Dravus St","15_W_Dra
"47.6538875432776","-122.37625660604","CMR-0013","15th Ave W & W Emerson St","15_W_Eme
"47.6685143331236","-122.376214714437","CMR-0010","15th Ave NW & NW Market St EW","15_
```

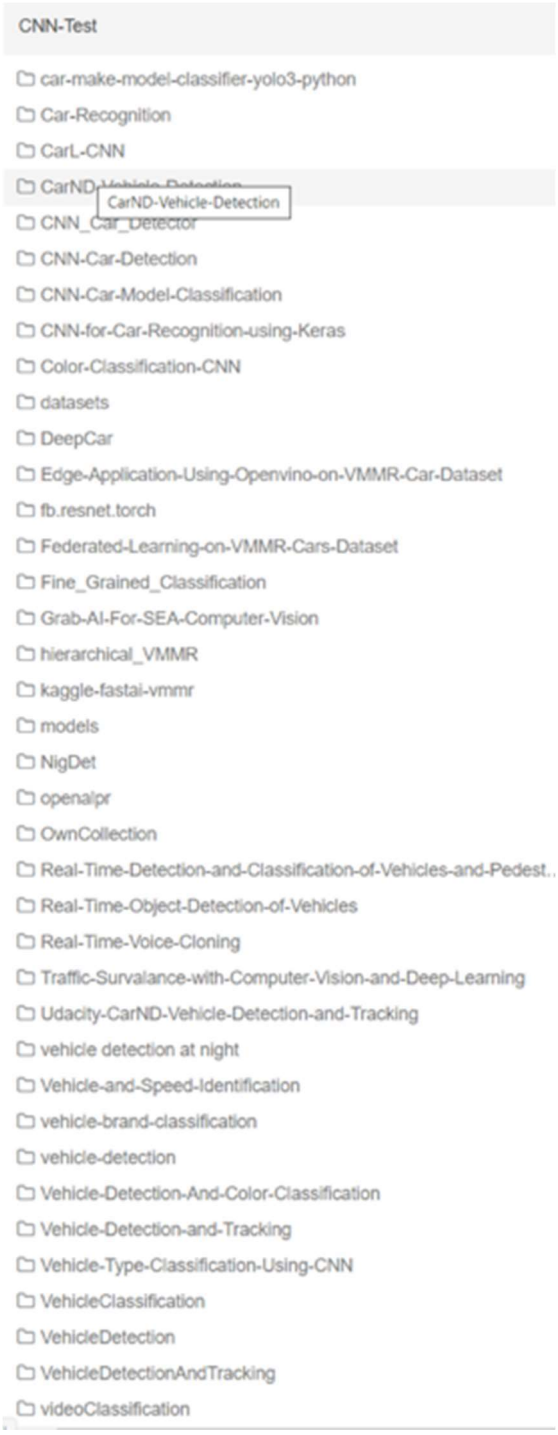
A very small script made it possible to parallelize the effort with an extremely small compute footprint. This resulted downloading a JPG to a folder

```
val cameras = cams.rdd.collect.iterator;

// define the tasks
val tasks = for (i <- 1 to numJobs) yield Future {
  // do something more fancy here
  watchCamera(ec,cameras.next).onComplete {
        case Success(result) => println(s"result = $result")
        case Failure(e) => e.printStackTrace
    }
  }

// aggregate and wait for final result
val aggregated = Future.sequence(tasks)
Await.result(aggregated, 600.seconds)
```

There are now **1,981** cameras watching for my car every minute of every moment of every hour for $20/bucks a day.



My Databricks sandbox instance has a large number of turnkey github projects with datasets and models either pre-built or just need to be trained once. These projects work with camera images or real-time video feeds.



I have the largest known datasets on vehicle recognition uploaded and ready.

Vehicle Make and Model Recognition Dataset (VMMRdb)

Overview

Despite the ongoing research and practical interests, car make and model analysis only attracts few attentions in the computer vision community. We believe the lack of high quality datasets greatly limits the exploration of the community in this domain. To this end, we collected and organized a large-scale and comprehensive image database called VMMRdb, where each image is labeled with the corresponding make, model and production year of the vehicle.

Description

The Vehicle Make and Model Recognition dataset (VMMRdb) is large in scale and diversity, containing 9,170 classes consisting of 291,752 images, covering models manufactured between 1950 to 2016. VMMRdb dataset contains images that were taken by different users, different imaging devices, and multiple view angles, ensuring a wide range of variations to account for various scenarios that could be encountered in a real-life scenario. The cars are not well aligned, and some images contain irrelevant background. The data was gathered by crawling web pages related to vehicle sales on craigslist.com, including 712 areas covering all 412 sub-domains corresponding to US metro areas. Our dataset can be used as a baseline for training a robust model in several real-life scenarios.

Cars Dataset

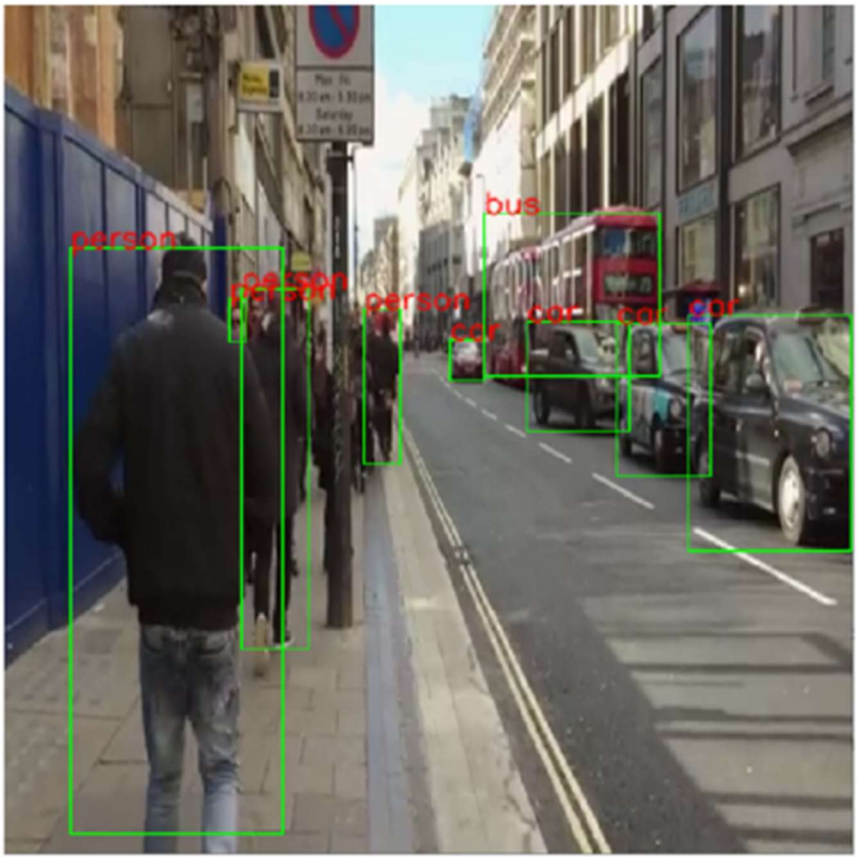
Overview

The *Cars* dataset contains 16,185 images of 196 classes of cars. The data is split into 8,144 training images and 8,041 testing images, where each class has been split roughly in a 50-50 split. Classes are typically at the level of *Make, Model, Year*, e.g. 2012 Tesla Model S or 2012 BMW M3 coupe.

The Comprehensive Cars (CompCars) dataset

By simply passing in a JPG, the following output is available. I also have datasets with trained models for use at **night time, traffic cameras, and aerial views.**

Fig. 2. Sample images from Chen dataset [2]. Some images are suffering from noise and brightness constancy.




设备编号:HZTCBB0026 时间:2018-07-24 08:31:47.143
White Rear wagoon
White Rear saloonCar
Grey Rear saloonCar
Green Front passengercar
White Rear suv
White Rear saloonCar
Gray Rear saloonCar
Red Rear saloonCar

ES.EPURE

1	2	3	4
			
Hyundai Azera Sedan 2012, prob: 0.99	Hyundai Genesis Sedan 2012, prob: 0.9995	Cadillac Escalade EXT Crew Cab 2007, prob: 1.0	Lamborghini Gallardo LP 570-4 Superleggera 2012, prob: 1.0
			
BMW 1 Series Coupe 2012, prob: 0.9948	Suzuki Aerio Sedan 2007, prob: 0.9982	Ford Mustang Convertible 2007, prob: 1.0	BMW 1 Series Convertible 2012, prob: 1.0
			
Mitsubishi Lancer Sedan 2012, prob: 0.4401	Cadillac CTS-V Sedan 2012, prob: 0.9801	Chevrolet Traverse SUV 2012, prob: 0.9999	Bentley Continental GT Coupe 2012, prob: 0.9953
			
Nissan Juke Hatchback 2012, prob: 0.9935	Chevrolet TrailBlazer SS 2009, prob: 0.987	Hyundai Accent Sedan 2012, prob: 0.9826	Ford Fiesta Sedan 2012, prob: 0.6502

As noted in my notebook, you could approach plotting these occurrences when a 2012 Cyan Prius C is spotted using the "scheduling" problem of overlapping intervals over a timeline. The images can also be saved and used for training against pictures of my own for specifics. Once the vehicle is identified once, the area where it was seen can be narrowed down. Past trying to find my car, I don't know where the use of this goes from here except that its ability to help people like me is **extraordinary.** Rekor Systems, for example, is a **private company** using crowdsourced ALPR technology to build a similar network that all subscribers to benefit. They are only using license plate technology, however, and not the **many other possibilities present here.**

I got this far over the weekend and last night. At this rate, I'll have my car back soon. Can you imagine the story? **"Car thief picked the wrong car to steal. Microsoft employees leverage machine learning technologies and publically available cameras to track them down."**

I'm currently moonlighting on this sort of thing and intend to move this over to private subscription. The cost has been remarkably minimal.

I wanted to open this up to see whom might have interest in joining me with to aid in tracking down my car and perhaps brainstorm ways we can leverage this for a greater good.

Thanks!

Christopher

Hello!!!

Over the last four months, I've received an enormous amount of feedback from people throughout the company regarding my stolen car and efforts to use ML to locate it. Recently, it's been people following up who are very curious how the story

ended. **So, over the last couple weeks, I've taken some time each night to share my 5 week journey up to the very end!**

The 95% or better majority included people who read the story, found it fascinating, and wanted to help out. There were an inquiry of a potential public safety case study, numerous offers for resources ranging from data lake space to compute, and countless people wanting to get involved and help. What I learned was, out there in the world – across the globe – our **worldwide communities at Microsoft** have constituents who are excited, passionate, curious, and ready to dive in to contribute to our one large Microsoft. *This felt very empowering and was certainly a key component powering the late nights and long weekends over the next 4 weeks to make an idea become reality.* **THANK YOU!**

I was able to quickly become very effective and deploy a massively scalable vehicle search operation across nearly 4,000 cameras with multiple stages of inference. On the final week, I was working on the final abstractions for extension to real-time traffic cam processing at a modest frame rate spanning several hundred Seattle-based traffic cameras. I also ended up purchasing an Edge TPU (Google *Coral)* with support for Tensorflow Lite and relatively fast neural network training capabilities. This thing is pretty cool! Despite the fact it claims you must have a Linux environment to install, and **Windows for Linux Subsystem** definitely doesn't excel at USB support (*even vNext by which I allow my Laptop to annoyingly restart at the most undesirable time so that I can have this early lol),* with a little effort you *can* use a serial port and get the job done.

Two names I must provide a special mention, @Moses Olafenwa and @John Olafenwa, creators of ImageAI. Their popular Python library significantly accelerated my ability to be very impactful quickly by wrapping functionality into a one-stop shop library offering a very usable interface. To illustrate its value, whether I was looking at Object Detection or Prediction, ImageAI provided a one-line interface that automatically downloaded models and assisted with custom training cases. Thank you for creating such a valuable and useful library!!!

One angle I hadn't considered when sending my first email was the **Ethical / Responsible AI** perspective. There were a handful of people who pointed that out and placed me in touch with the Responsible AI folks to have a discussion. Provided the nature of what I was doing, very legitimately as it were, could easily be construed or abused for purposes now being highlighted pretty heavily in the media by Tiktok and other forms of AI-powered surveillance, *I kept my work in a private Github and did not publicize details any further*. Just to be clear, at no point were license plates or any form of personally identifiable metrics being processed. This project was limited solely on make/model/color classification of a vehicle at scale. **All Responsible AI principles were considered in their entirety and much thought/care was given as a result. Just to be safe since there was concerned raised behind the intent of this work, I chose not to follow up on offers to join the cause or contribute resources as to be sure not to perpetuate this at any scale, at least while I was trying to find my car.**

In all transparency, there are some elements of irony present. There is a strong possibility my perception of irony and the resulting ambiguity is solely based on my

current level of knowledge in the responsible/ethical AI subject matter. The remaining 5% of the responses I received, which could best be characterized as not too enthusastic and pointing out the *potential* ethical AI concerns, were not at all excited about the prospect of this idea and altogether. My first impression, propeled forward from the zeal of the large majority whom I could only view as embracing a growth mindset, was genuine conviction behind wanting to understand the facets surrounding decisions to embrace a particular area of research or bury it solely based on the fact it could be used for good or harm. I began wondering when faced with capabilities lending to good or harm, provided there is no real secret here, if taking a passive role and choosing not to have a voice really is responsible or ethical AI? Granted there are much more important considerations in the world today besides dwelling to heavily in this area.

In order to illustrate my irony, I want to reference this **2018 Surveillance Impact Report** released in May 2019 so that policy position can be established.

(Skip the following commentary if you're solely interested in the journey and whether I found my car)

2018 Surveillance Impact Report

CLOSED CIRCUIT TELEVISION "TRAFFIC CAMERAS"

SEATTLE DEPARTMENT OF TRANSPORTATION

http://www.seattle.gov/Documents/Departments/Tech/Privacy/CCTV%20Traffic%20Cameras_Final%20SIR.pdf

Note, SDOT is one of many entities throughout WA with cameras.

Purpose of Cameras

SDOT operates an extensive network of 210 Traffic Cameras across Seattle to help our Transportation Operations Center (TOC) detect and quickly respond to congestion, incidents, and other problems on the roads. The ability to observe traffic conditions across the City in real-time is a primary component of SDOT's Traffic Incident Management (TIM) program. TIM consists of a planned and coordinated multi-disciplinary process to detect, respond to, and clear traffic incidents so that traffic flow may be restored as safely and quickly as possible. Effective TIM reduces the duration and impacts of traffic incidents and improves the safety of motorists, crash victims and emergency responders.

The live streaming video from each of these cameras is also accessible to the public on our Traveler Information Map, and used by other city departments to understand current traffic conditions. The majority of workers surveyed in Seattle and Los Angeles indicated that they used the Internet to obtain traffic information for their commute trip (FHWA, 2005).

(Federal Highway Administration (FHWA), United States Department of Transportation (USDOT), 2005).

Surrounding Applicability to Government-Focused Criminal Investigation

Article 1, Section 2 – the US Constitution is the supreme law of the land;

Article 1, Section 7 - Invasion of Private Affairs or Home Prohibited

Article 1, Section 32- "A frequent recurrence to fundamental principles is essential to the security of individual right and the perpetuity of free government."

3. Context for Seattle: The above means essentially:

You cannot simply 'surveil everything' in the hopes of finding a criminal (or even worse, someone you simply "don't agree with"). That is called 'guilty until proven innocent' and has been overturned time and time again in our system of laws by courts and legislators at every level. The Bill of Rights has protected the 4th Amendment concept of 'Innocent until Proven Guilty' and 24-7 surveillance of **any** sort flies in the face and openly defies this most basic law.

You cannot 'surveil' public assemblies, protests, or similar gatherings, most especially with facial recognition, phone network/bluetooth data capture or public video recordings and/or microphones without again, violating the above basic constitutional principles – otherwise known as "laws" (US and WA).

You cannot store data simply according to 'policy', or come up with what you believe adequate controls may or may not be, and then implement them without complete transparency and public input, including that of the City Attorney's office, elected officials and arguably most important, THE PUBLIC. I believe this effort you have begun to solicit feedback is a good start, but there's a long way to go and this is only the very beginning, rest assured.

Finally, you cannot pay lip service to these previous paragraphs by not actively doing them yourself, and then simply turn around and receive/use/retain the data anyway through other means – that is, you cannot obtain the data from the NSA's Fusion Center already located in downtown Seattle, or the FBI, or TSA, DHS, or increasingly rogue agencies like ICE – all of these still break the law, plain and simple.

Seattle Specific Data Practices to Protect Freedom

Specific technologies being discussed in this public outreach:

1) SDOT LPR's.

Positive – the data is stated as being deleted immediately after a transit time calculation;
Positive – the data is stated as only being available to SDOT personnel after relay from WSDOT, with individual identifying license plates not part of that incoming data;
Positive – stated purpose – facilitate effective and efficient traffic management within the Seattle city limits.

Seattle City Government Sentiment Surrounding WSDOT Surveillance Practices

SDOT LPR's - COMMENT for Submission/consideration:

a) It is unclear how long WSDOT is retaining this data for handoff to SDOT and Seattle generally – even if SDOT deletes it nearly immediately after a calculation/use, can they go back and re-retrieve it later? The answer should be NO, and simply that WSDOT is doing the same thing at minimum – deleting the data almost immediately after said calculation too (I recognize this latter is beyond SDOT's control, however, certainly as the biggest city in the state, Seattle would have major influence on these policies and procedures were you to weigh in and state clear policy positions).

b) It is also unclear what the statement 'travel time calculation' precisely means for these purposes. Is it just me driving through downtown and getting spotted if I go by any of these cameras/devices? Assuming the answer is yes, when is the 'timeout' – 1 minute if not seen by another camera? 5 minutes? When and how quickly does the 'calculation' occur (so that I know purportedly the data is then "immediately deleted" as you say?

c) It is also unclear if anyone else working for the City of Seattle has access to this WSDOT data (and if so, for how long, in what capacity, at what level of detail, etc.) – say, the SPD, City Attorney's office, or? So maybe SDOT isn't "surveilling" anyone within the normal meaning of the term given the safeguards noted in the policy PDF, but certainly the SPD have far different reasons for using this data, and most (if not all) of them are far removed from simple data calculations, and include direct data review to carry out those tasks?

ACLU Response to Seattle's Surveillance Impact Report

ACLU
AMERICAN CIVIL LIBERTIES UNION of WASHINGTON

October 24th, 2018

RE: ACLU-WA Comments Regarding Group 1 Surveillance Technologies

Dear Seattle IT:

On behalf of the ACLU of Washington, I write to offer the ACLU-WA's comments on the surveillance technologies included in Group 1 of the Seattle Surveillance Ordinance process. We are submitting these comments by mail because they do not conform to the specific format of the online comment form provided on the CTO's website, and because the technologies form groups in which some comments apply to multiple technologies.

These comments should be considered preliminary, given that the Surveillance Impact Reports for each technology leave a number of significant questions unanswered. Specific unanswered questions for each technology are noted in the comments relating to that technology, and it is our hope that those questions will be answered in the updated SIR provided to the City Council prior to its review of that technology.

3. Closed Circuit Television "Traffic Cameras" (SDOT)

As with the other two camera technologies, the crux of concern around these traffic cameras relates to limiting their use to specific purposes, enshrining in statute protections against invasion of privacy and general data collection, and limiting data sharing. It would be helpful to see the SDOT camera control guidelines referenced in the SIR, as well as to make clear in a policy applicable specifically to these cameras, what data will be deleted when (Section 5 appears to contain several different retention policies). Additional questions that an updated SIR should answer are as follows:

- The current SIR does not reference specific camera vendors and models—these would be helpful to have.

Same Story from my first email except for one twist <u>no one considered.</u>

Toll and traffic safety cameras can't be used to fight crime. Washington Legislature should fix that

BY THE NEWS TRIBUNE EDITORIAL BOARD
JANUARY 26, 2019 02:40 PM , UPDATED JANUARY 29, 2019 12:14 PM

Imagine you're the unlucky victim of a car prowl but the lucky recipient of a hot tip just minutes after your stuff was stolen. A credit card company alerts you that a fraudulent transaction was attempted with your card at the Tacoma Narrows Bridge tollbooth.

So you do what any take-charge crime victim would do: You notify police, then quickly contact the Good to Go tolling office for information about the vehicle that was blocked from using your card.

That's when the hot tip goes ice cold. Good to Go isn't authorized to share any toll camera data for crime-fighting purposes. Not with you, and not with detectives. Sorry, Charlie, but the cops already know the drill.

https://www.thenewstribune.com/opinion/editorials/article225098710.html

Summary Points Framing Irony

What we've learned

- For purposes of championing *"Traffic Management"*, a vast surveillance network of traffic cameras is installed throughout Seattle (*and the State)*.
- There is massive push-back and concern having this technology in the Government's hands will result in other government agencies using it for purposes that infringe against the individual people's civil liberties and individual freedom.
- An agreement has been reached requiring very specific controls to be in place in order to prevent abuse by government agencies. One such control is "*near immediately*" deleting camera data as it occurs.
- A position of complete transparency has been embraced so that all policies, discussion, commentary, and access to the cameras themselves are all made available to the public domain.

Set the Stage for our Ironic Plot

Yakima Herald-Republic

SEATTLE — Could images from red-light cameras have led Seattle police to a pair of kill

The ubiquitous cameras are mounted a short distance from the Pioneer Square interse
Westbrook, 21, was fatally shot in April by a gunman in a passing car. Other cameras ar
too far from where Justin Ferrari, 43, was killed while driving with his family about a mo

Detectives wonder if the cameras may have captured images — a fleeing car or a gunm
that could have helped investigators.

But police are barred by state law from accessing and using images from the controversial cameras for anything other than traffic enforcement. Still, the knowledge that the cameras could provide vital clues in the unsolved slayings is equal parts tantalizing and frustrating for detectives.

Washington is among the few states that bar police from using images from red-light cameras in criminal investigations. The way the 2005 law is tailored, even if a homicide, abduction or any other serious crime occurs within full view of the cameras, the images cannot be used by police, said King County Senior Deputy Prosecutor Don Raz.

Now Let's Ask the Questions

1.) Suppose my vehicle was identified from a civilian happening to watch public traffic cameras. If police acquire information or images from a non-government entity, could either that information or representation of the image itself be used to further a criminal investigtion? In other words, if I identified my car from a traffic camera and even know where its parked, is it a non-starter to for police to help since it originated from a traffic camera?

2.) Consider the last paragraph above regarding abductions above. Here's a story…around 4pm, Jack always opens the traffic cameras before leaving work as to pick which way he'll go home. Someone kidnapped a child and, while he didn't realize it at the time, he remembered just a bit later after hearing the Amber Alert. By some stroke of luck, Jack was sharing his screen in a Teams meeting that was recorded and it consequently captured the abduction in progress. Jack sends a screnshot of his computer observing the Seattle traffic camera online. There is an abandoned building across the street and the screenshot shows the child being taken inside. Do police rescue the child? Can the abductor be arrested if the premise of the investigation is against the law?

3.) Okay, Seattle, you did it. You now have your traffic cameras in place everywhere and have established by law that these cameras are your "ball" and you will not be sharing with anyone else, except the public for whom you will be completely transparent and offer full access. What happens next? Private companies and individuals begin consuming this data at scale! This was never considered! Eventually, someone comes along who – like the news article above

points out – was a victim of a crime and learns the traffic cameras are not availble for use. Instead of the trail going cold, this individual decides to use the cameras to find his car. Eventually, the cameras are used to help prevent crime and offer real-time alerts to anyone. Data brokers begin archiving live feeds and selling the image as datasets for whichever interest you might have.

So, here we sit at the forefront of this type of paradigm, and the position I've heard advocated is to passively ignore the technology so that we are not perpetuating potential bad use of AI. My personal opinion, which admittedly is likely ambiguous and less informed, believes embracing a "*growth mindset*" in this space means we take a more active role in shaping that landscape. Avoiding controversial technology altogether is the same circular reasoning around any abstract concept whereby potential for good or evil may occur. There are some really difficult questions here that have not been answered and, based on the literature, not once even considered or called out as a risk.

Consider some of the outcomes. Let's say Amber Alerts. If police are unable to use traffic cameras to save a child's life and the cameras are publically available. Taking a stance or position in this area could mean a child's life is saved. It could mean leading an example whereby the data isn't commoditized and commercialized by setting some precedent as a use for the greater good. The truth is, it took me about 3 hours to make good progress. Others will figure it out and use it for their own purposes, for whichever good or evil purpose that might be. It's very likely much harm will come from this fact before some action is actually taken to remove these cameras from public use. In the global conversation surrounding responsible and ethical AI for good, what do we do to create impact in the world and shape these outcomes for good at a time when the framers behind making this possible had not anticipated such an outcome?

I don't know the answer to the above questions, I

Now that we've had that discussion…let's talk about my car!!!!!!!!!!!

My Journey Forward….

If you haven't read or missed the first installment to this saga, my vehicle was stolen right out of my driveway in late March! It was a 2012 Toyota Prius C with over 150k

miles. It wasn't until receiving a toll charge a couple weeks after it was stolen and learning that "Good to Go" would not provide me video of my vehicle that I began to wonder about all the cameras around us throughout Seattle. I discovered there were thousands of publically accessible cameras postted on the web that cannot be used by law enforcement and seem to exist for public use, primarily for traffic. I reasoned that I could have scripts watch all these cameras and alert me when my Toyota Prius C appeared so that I could alert the police and get my car back. Within a few short weeks, my project had concluded the second week of May…

Camera Acquisition

Towards the end, there were over **4,000** publically available web cams in use for traffic or other purposes cataloged and grouped into json structures. Every web cam published a JPG to the internet at various intervals, some also providing live streams as well. Most sources had some underlying json document like below which contained a dictionary of all URLs and made it very trivial to quickly build a large cache by just looking at the payloads from your browser's "network" tab.

master ▾ PriusWatchML / GreaterSeattle315.json

 cchighman Add files via upload

1 contributor

3917 lines (3917 sloc) | 109 KB


```
[
 {
    "id": 9580,
    "CameraLocation": "null|B|47.59354|-122.329735|2|I-90",
    "CameraOwner": "",
    "Description": "",
    "DisplayLatitude": 47.59354,
    "DisplayLongitude": -122.329735,
    "ImageHeight": 249,
    "url": "https://images.wsdot.wa.gov/nw/090vc00202.jpg",
    "ImageWidth": 335,
    "IsActive": true,
    "OwnerURL": "",
    "Region": "NW",
    "SortOrder": 1300,
    "Title": "I-90 at MP 2: 3rd Ave S"
 },
 {
    "id": 1434,
    "CameraLocation": "null|B|47.5945|-122.32915|2|I-90",
    "CameraOwner": "",
    "Description": "",
    "DisplayLatitude": 47.5945,
    "DisplayLongitude": -122.32915,
    "ImageHeight": 249,
    "url": "https://images.wsdot.wa.gov/nw/090vc00207.jpg",
    "ImageWidth": 335,
    "IsActive": true,
    "OwnerURL": "",
    "Region": "NW",
    "SortOrder": 1300,
    "Title": "I-90 at MP 2: 4th Ave S, EB"
 },
 {
    "id": 9581,
    "CameraLocation": "null|B|47.592767|-122.323958|2|I-90",
```

Methodology

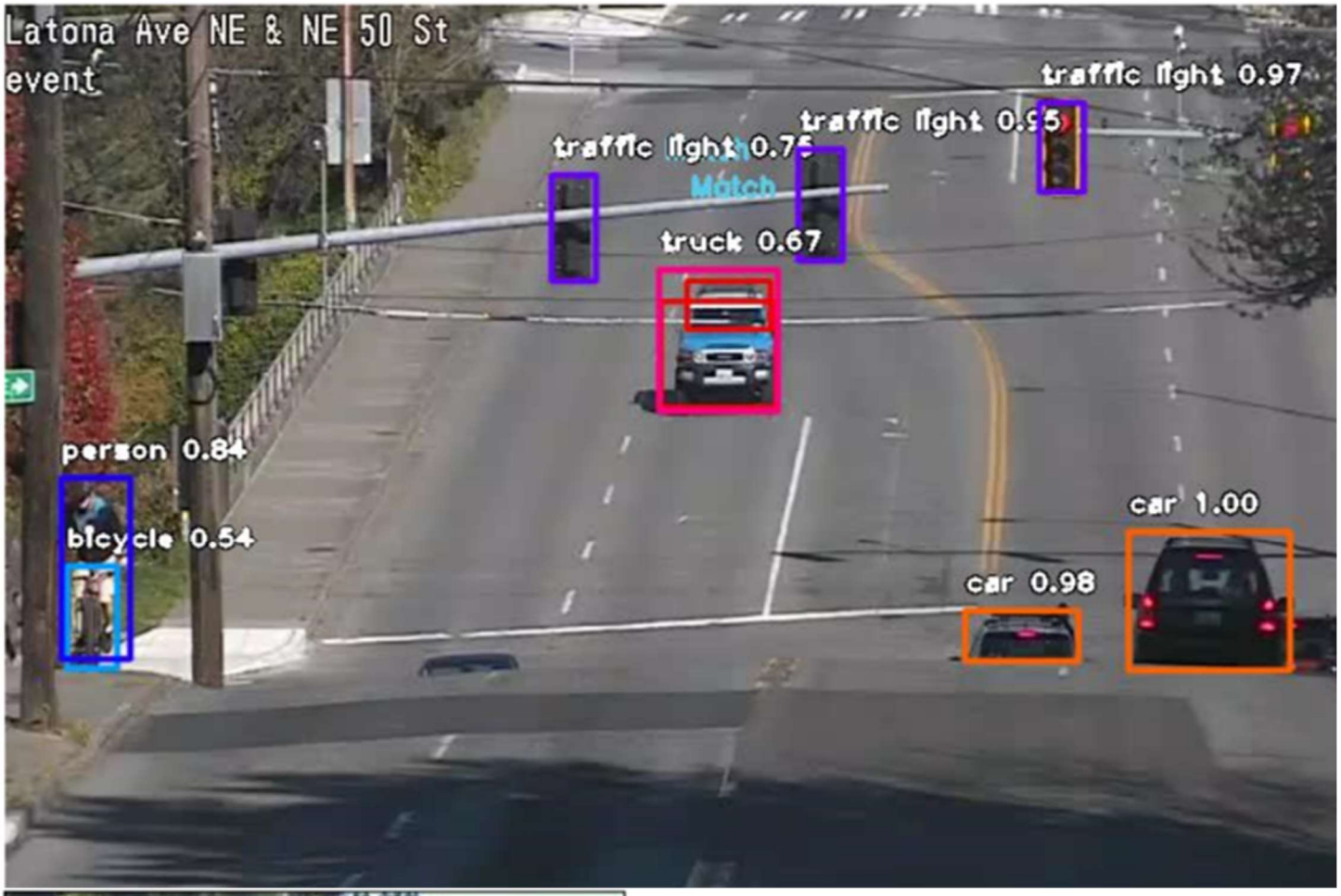
Getting this formula correct was tricky. I was lucky enough to find a few examples from Github that were quite actionable but still far from perfect.

The workflow was rather simple but the challenge was in fine-tuning the classification model and performing inference at scale.

1.) Python script looped through an assigned list of JSON objects containing cam metadata.
2.) Image was acquired from URL and a hash was computed in order to determine if it was a duplicate.
3.) Object Detection using pre-trained Yolo v3 model to identify cars within the provided image and extract them to a numpy array.
4.) Eucilidean distance method of contour color identification to verify a Toyota Prius color palate before performing inference.
5.) Finely-tuned convolutional neural network with transfer learning from pre-trained ResNet-50 CNN to classify vehicle type as either Prius or Vehicle.

Object Detection

ImageAI made object detection a piece of cake. Here's an example of what Yolo does out of the box to a provided image:



In my case, I was only concerned with the "car" object detection. Each of the boxes drawn around the detected objects were extracted into an array.

Vehicle Classification

ImageAI also made it a piece of cake to build prediction models and perform inference to cherry pick which vehicle was a Toyota Prius C from a provided image.

Here's an example of real example of success: a reddish box surrounds a Toyota Prius C having the word "Match" at the top.

Tech Choices

It was my goal to use Azure technology where ever possible. On my personal subscription, I began with **Azure Databricks** but due to Covid, GPU resources were very limited. Limited as in zero inventory. This is by far the best route to go, however, for this purpose. When you combine multi-core GPU availability for distributed training and inference along with the opportunity to use Apache Spark "Pandas UDF" functions to distribute execution context workloads which do not need to be on a driver, it's covering every need for this project to be successful. As we'll see below, I was having to do a bit of extra work and manual effort in order to replicate the same behavior. Ultimately, though, as a personal consumer, Azure Databricks would likely be cost prohibitive for me over time.

Azure Cognitive Services was very promising. It's classifier had a very simple interface to use for training, I believe it was the **Custom Vision** offering. Several team members from Azure Cognitive Services had reached out to me as well when my first email went out. I was excited about using **Azure Custom Vision** with initial results showing great promise. The UI was very friendly but also powerful. They provided a very clean way to link a web user with a wrapped classifier training experience. I didn't keep course here, though, because prediction cost per API invocation at my scale with no foreseeable end date in mind was well outside my personal budget. There was some talk from a PM in this space around the time my first email came out to use this project as a Public Safety case study.

With **Azure Containers**, I was able to quickly spin up an instance as either a simple "nginx" container or one more basic. Unfortunately, to use this container for my purposes, I really needed to have SSH access and not use the provided web interface. Being immutable and based on an image, this was really not ideal which is likely why SSH isn't available in the first place.

Considering ~4,000 cameras x 1 per minute x 60 minutes in hour x 24 hours ~= 5,760,000 predictions per/day, my costs would go up fast paying per prediction.

After various phases of using a number of local devices by which I could run a python script and first discovering Kaggle offers a GPU to use for training in a managed notebook, I eventually learned about Google Colab.

Google Colab allowed me to spend $10/month and run a number of different notebook instances for nearly 24 hours using a **Tesla 100** GPU for Compute. **Azure Notebooks** were brand new at this point and hadnt achieved feature parity with Colab at this point. I quickly learned, however, Google Drive was optimized heavily for reads and **not writes**. For this reason, when I was only saving my results to a JSON file on disk, I'd come to discover they were never actually written because Google Drive was backed up to infinity.

I ended up creating a small Azure Function and attached an HTTP trigger for the purpose of recording matches. Using VSCode with Python for Azure function deploys was broken at the time so it provided an opportunity to raise awareness back to maintainers: https://github.com/microsoft/vscode-azurefunctions/issues/2071. They marked it as a P1 bug and promptly resolved the issue.

Initial Inspiration

https://github.com/spectrico/car-make-model-classifier-yolo3-python

My initial inspiration was based on this Github project above which offered a lite version of their classifier and paid access to their classification APIs through a third-party site. The lite classifier showed promise but overall had very poor accuracy. I wanted to leverage other tech as much as possible so I spent several hundred dollars on initial prototypes using the paid version with per API invocation payment model. Even the paid version was sub-par for my needs and it began getting expensive

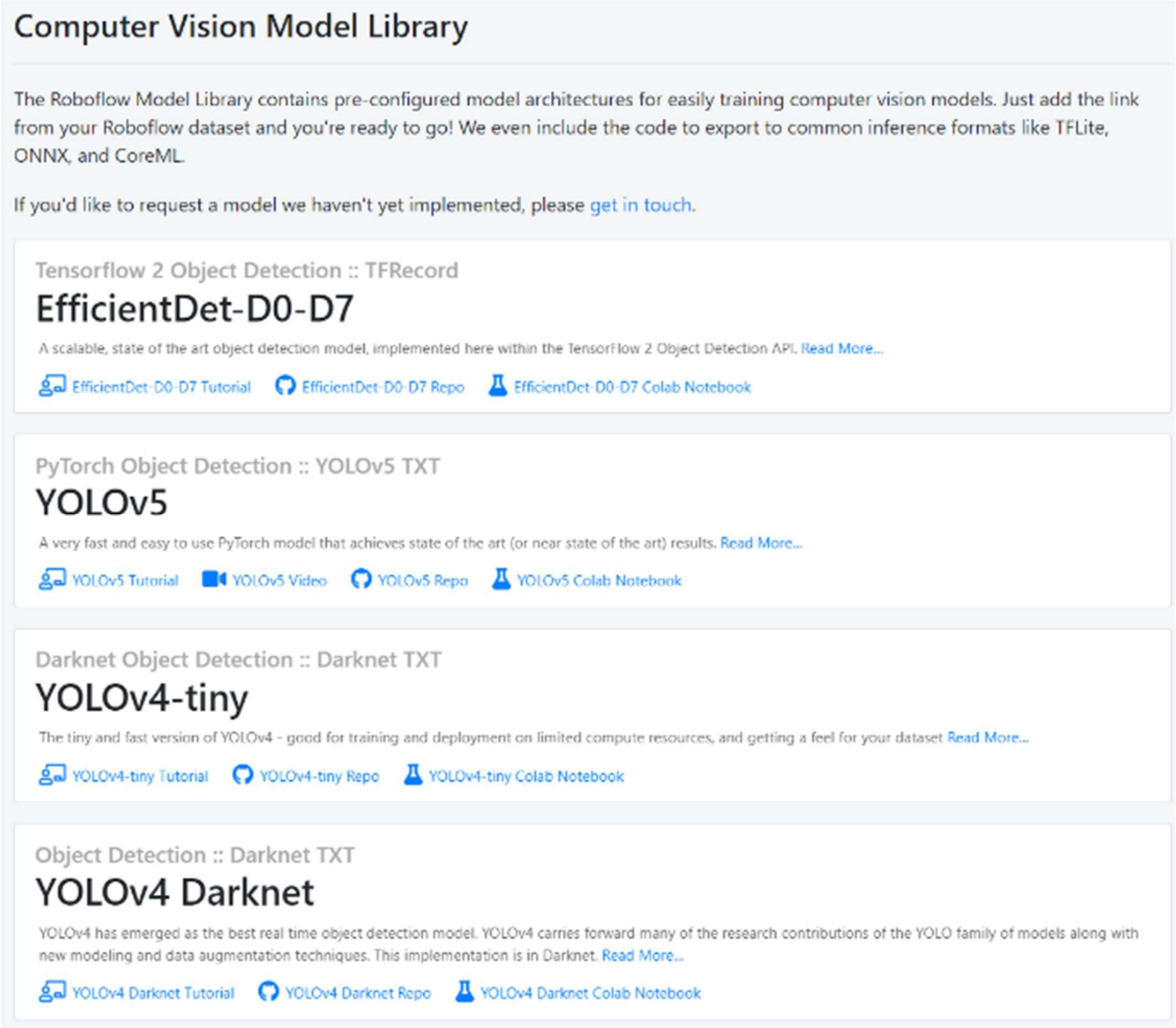
so I realized I was going to need to train my own deep learning model to get the job done.

The other dealbreaker, the above project used OpenCV DNN version of yolov3. OpenCV doesn't support GPUs/CUDA officially, however, you can recompile it with extensions to support GPUs.

First Step – Object Detection

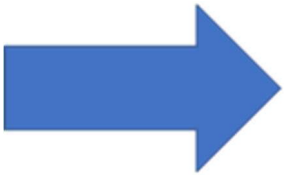
Object Detection, as we saw above, is principally concerned with taking an image and returning back a list of detected objected to a certain degree of confidence. There are a number of pre-trained object detection libraries out there which are ready for use. When I began this project, I was using YoloV3. When performing object detection, its typically required for all images considered for detection have the same dimensions by which the model had been trained. 220x220, for example. **ImageAI** implicitly performs image resizing and cleanup so that consumers don't even have to know it's a thing.

Most pre-trained object detection datasets have various trade-offs between accuracy and performance in terms of frames per second. One of the values behind using **ImageAI,** the steps needed to download these various object detection models and get them going is completely wrapped for you. You can switch back and forth between object detection models as parameters and the rest is done for you.



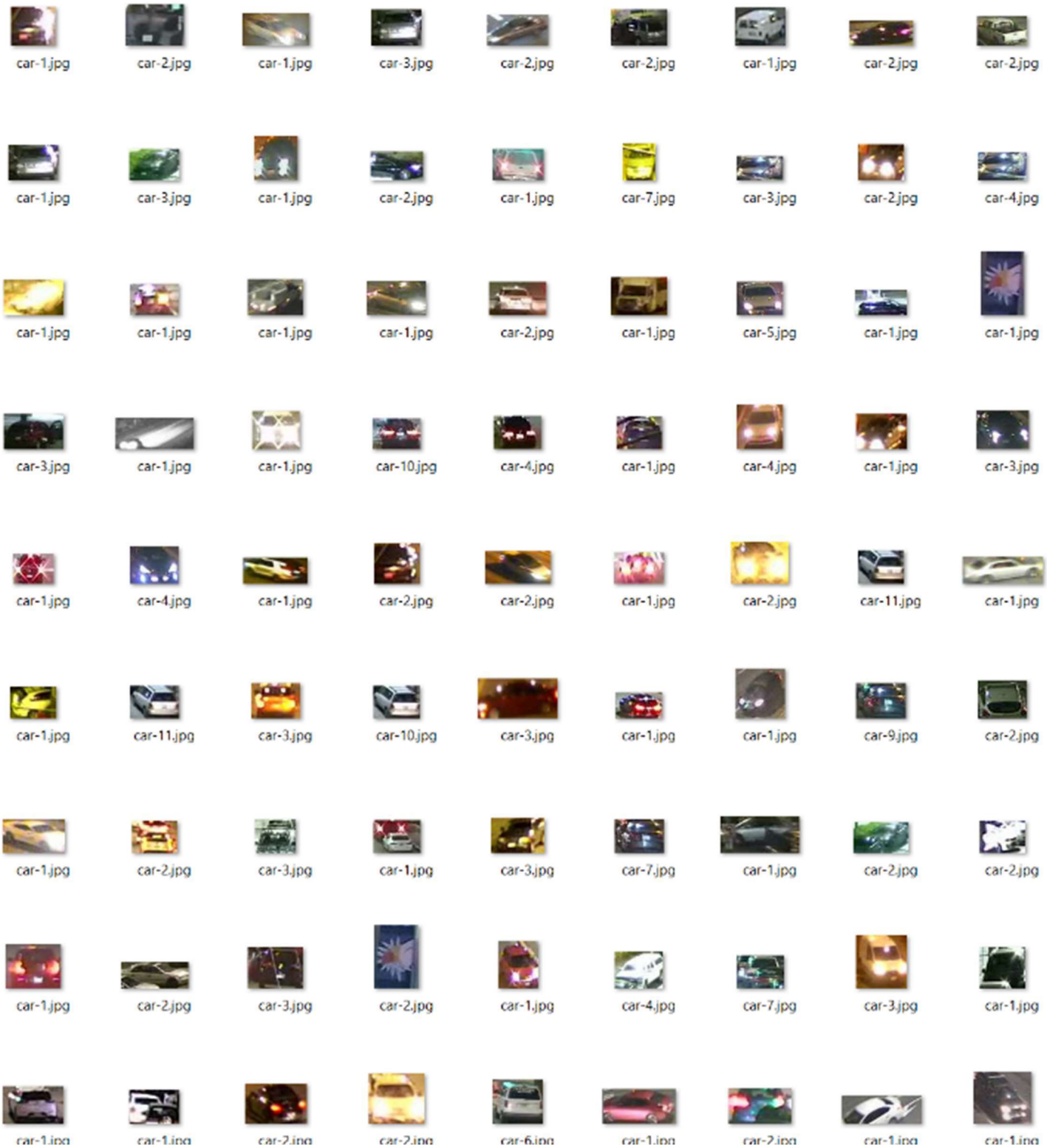
https://models.roboflow.ai/object-detection

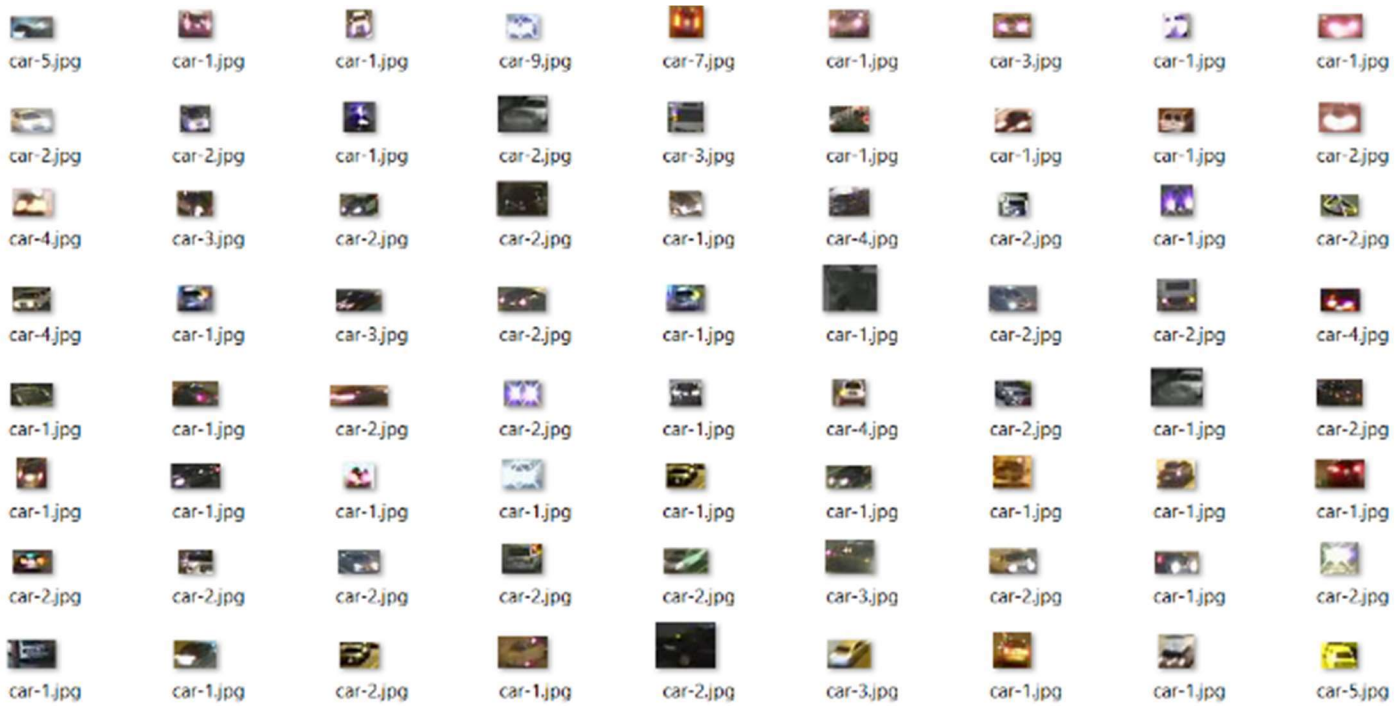
It's worth pointing out here this picture below is actually the result of *classification* and not *object detection*. With *object detection*, I would expect to see bounded boxes surrounding at least 75% of the vehicles in the below image. All of those would be extracted like shown below and saved according to your preference. It's needed for the next step.



Examples of vehicles extracted from photos

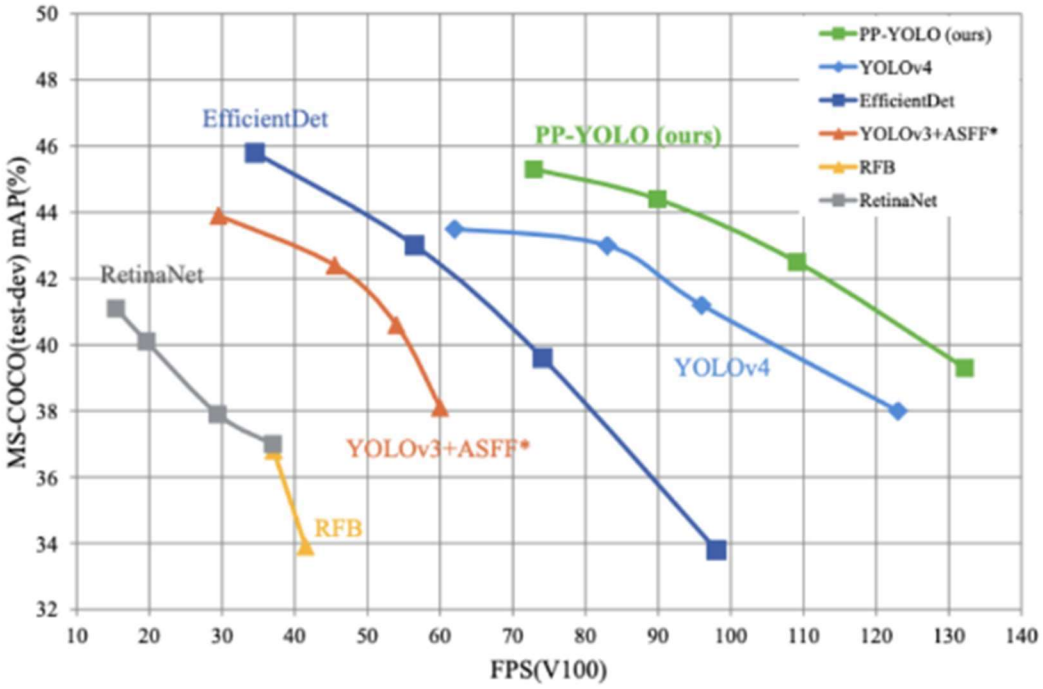
These images below are real extractions from traffic cameras. Some of them are quite small in size. It's accuracy is very impressive. Once this extraction has occurred, I'm in the best shape to reason further about the extracted image.


car-1.jpg car-2.jpg car-1.jpg car-3.jpg car-2.jpg car-2.jpg car-1.jpg car-2.jpg car-2.jpg
car-1.jpg car-3.jpg car-1.jpg car-2.jpg car-1.jpg car-7.jpg car-3.jpg car-2.jpg car-4.jpg
car-1.jpg car-1.jpg car-1.jpg car-1.jpg car-2.jpg car-1.jpg car-5.jpg car-1.jpg car-1.jpg
car-3.jpg car-1.jpg car-1.jpg car-10.jpg car-4.jpg car-1.jpg car-4.jpg car-1.jpg car-3.jpg
car-1.jpg car-4.jpg car-1.jpg car-2.jpg car-2.jpg car-1.jpg car-2.jpg car-11.jpg car-1.jpg
car-1.jpg car-11.jpg car-3.jpg car-10.jpg car-3.jpg car-1.jpg car-1.jpg car-9.jpg car-2.jpg
car-1.jpg car-2.jpg car-3.jpg car-1.jpg car-3.jpg car-7.jpg car-1.jpg car-2.jpg car-2.jpg
car-1.jpg car-2.jpg car-3.jpg car-2.jpg car-1.jpg car-4.jpg car-7.jpg car-3.jpg car-1.jpg
car-1.jpg car-1.jpg car-2.jpg car-2.jpg car-6.jpg car-1.jpg car-2.jpg car-1.jpg car-1.jpg



Relative Performance of Popular Object Detection Methods

There are object detection datasets spanning a range of use cases. Here's a chart below identifying relative performance among some of the more popular datasets.



[YOLOv4: Optimal Speed and Accuracy of Object Detection]

Second Step - Classification

Classification of a custom object is by far one of the biggest challenges to overcome. In order for CNN to identify the object in question, a very representative dataset should be provided for training. If beginning from scratch, deep learning for your model could take a very, very long time. Fortunately, there are ways by which we can share the intelligence of other models previously built so that we start already having a solid base understanding of patterns and the associated pre-trained weights.

1. Transfer learning

Transfer learning is a popular method in computer vision because it allows us to **build accurate models in a timesaving way** (Rawat & Wang 2017). With transfer learning, instead of starting the learning process from scratch, you start from patterns that have been learned when solving a different problem. This way you leverage previous learnings and avoid starting from scratch. Take it as the deep learning version of Chartres' expression 'standing on the shoulder of giants'.

https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

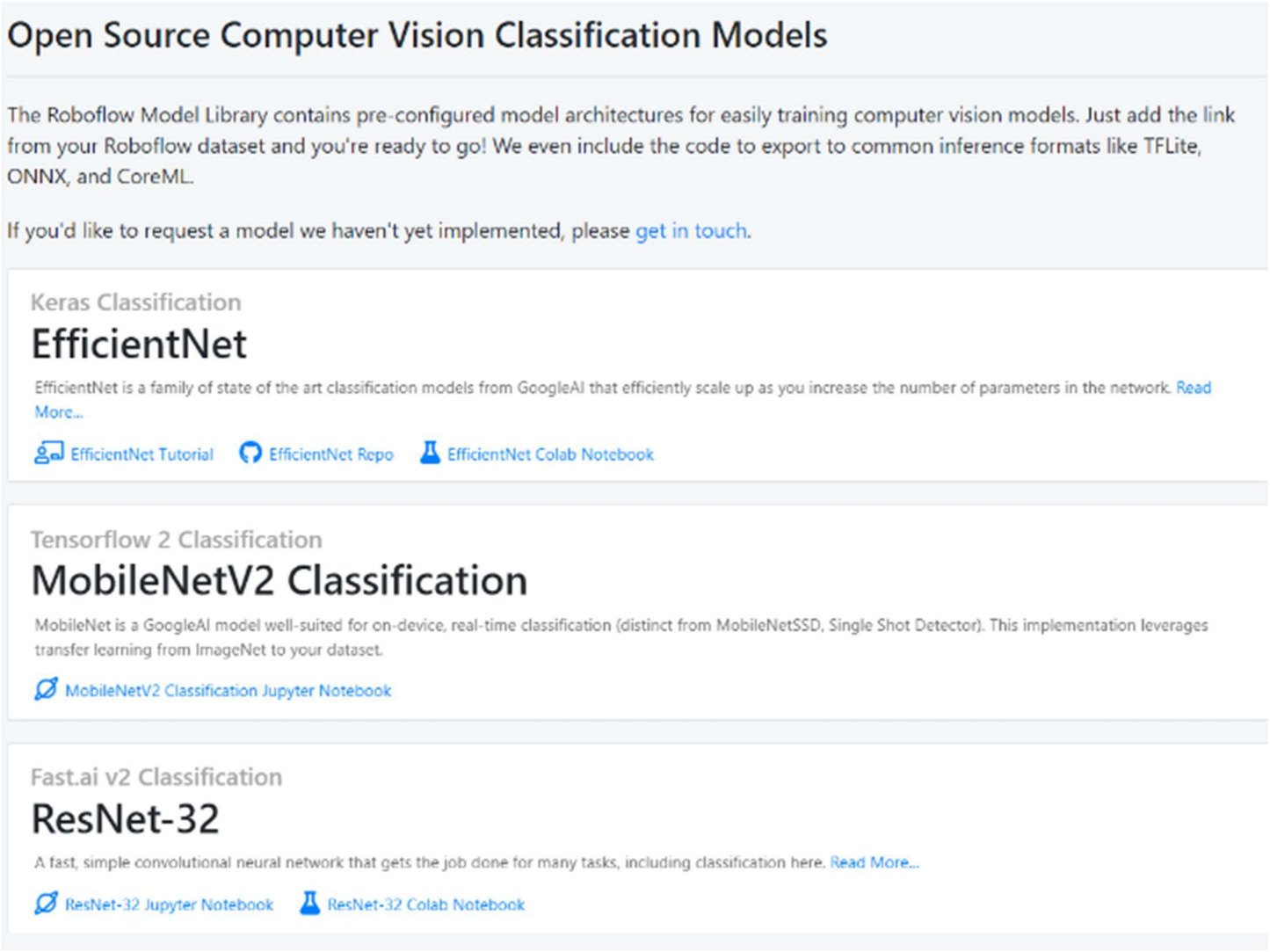
Provided the below explanation, using a pre-trained dataset for a classification task would be providing the "*convolutional base*". **ImageAI** has parameters in its custom training API to specify whether you are transfer learning or possibly continuing where you left off while training another dataset. In my case, I'm starting with nothing and definitely need a convolutional base. There are a number of considerations surrounding which type of dataset you'd want to leverage for this purose.

A typical CNN has two parts:

1. **Convolutional base**, which is composed by a stack of convolutional and pooling layers. The main goal of the convolutional base is to generate features from the image. For an intuitive explanation of convolutional and pooling layers, please refer to Chollet (2017).
2. **Classifier**, which is usually composed by fully connected layers. The main goal of the classifier is to classify the image based on the detected features. A fully connected layer is a layer whose neurons have full connections to all activation in the previous layer.

After much consideration, I went with **ResNet-50**. ResNet-50 happens to do a great job at solving the "*vanishing gradient*" problem other pre-trained datasets for image classification do not solve. As a result, those other pre-trained datasets might have restrictions around how deep or fully connected their network can be.

https://medium.com/analytics-vidhya/understanding-and-implementation-of-residual-networks-resnets-b80f9a507b9c
Deep Residual Learning for Image Recognition' research paper from Microsoft Research
https://www.kaggle.com/keras/resnet50
https://pytorch.org/hub/pytorch_vision_resnet/



https://models.roboflow.ai/classification
https://developer.nvidia.com/deep-learning-performance-training-inference

Transfer Learning in Action with ImageAI

Assuming you have already prepared your dataset, you would be ready to create your new classifier by using transfer learning! We will go deeper into the dataset I built for my Prius just after this. Using **ImageAI**, this is literally everythign required to make the magic happen. In the below case, I'm using transfer learning to create a new model while attempting to use a previous model as a "base". Notice the parameters below for

things like *save_full_model, enhance_data,* and *transfer_from_model*. These are the knobs you're able to turn to make everything work like magic.

Tranfer Learning from Previous Prius Detection Model -- specializing against traffic camera ima

In [8]:
```
import os
from imageai.Prediction.Custom import ModelTraining

model_trainer = ModelTraining()
model_trainer.setModelTypeAsResNet()
model_trainer.setDataDirectory("/content/cars2")
model_trainer.trainModel(num_objects=2,
                         num_experiments=100,
                         save_full_model=True,
                         transfer_from_model="/content/drive/My Drive/Colab
                         enhance_data=True,
                         batch_size=32,
                         show_network_summary=True)
```

https://github.com/cchighman/ImageAI-YOLOv3-Fine-Tuning-Vehicle-Classification/blob/master/ImageAI_Fine_Tuning_Resnet50_for_Vehicle_Classification_based_on_YOLOV3_Object_Detection.ipynb

While **ImageAI** does a good job of handling this next portion for you, for illustrative purposes I wanted to make sure it's mentioned. In order to carry out *transfer learning*, typically your convolutional base would be *frozen* so that nothing can change its underlying representation.

1. Feature Extraction: Use the representations learned by a previous network to extract meaningful featu samples. You simply add a new classifier, which will be trained from scratch, on top of the pretrained you can repurpose the feature maps learned previously for the dataset.

   You do not need to (re)train the entire model. The base convolutional network already contains featur generically useful for classifying pictures. However, the final, classification part of the pretrained mod to the original classification task, and subsequently specific to the set of classes on which the model

2. Fine-Tuning: Unfreeze a few of the top layers of a frozen model base and jointly train both the newly-a classifier layers and the last layers of the base model. This allows us to "fine-tune" the higher-order fe representations in the base model in order to make them more relevant for the specific task.

https://www.tensorflow.org/tutorials/images/transfer_learning

I followed the method of training my classifier until my accuracy and loss was very low without any real hope of further improvement. This worked perfectly so long as I had a well-balanced dataset. For example, if I was building a classifer capable of predicting two different classes and I didn't provide nearly enough images for training, there is a possibility you may observe unexpected results while training and afterwards. For my use case, I only wanted two classes. It's either a "vehicle" or it's a "prius".

```
Epoch 98/100
247/248 [============================>.] - ETA: 0s - loss: 5.8305e-04 - acc: 0.9999Epoch 1/1
107/248 [===========>..................] - ETA: 10s - loss: 0.0406 - acc: 0.9968
Epoch 00098: val_acc did not improve from 0.99679
248/248 [==============================] - 113s 456ms/step - loss: 6.0056e-04 - acc: 0.9999
Epoch 99/100
247/248 [============================>.] - ETA: 0s - loss: 3.2935e-04 - acc: 1.0000Epoch 1/1
107/248 [===========>..................] - ETA: 10s - loss: 0.0403 - acc: 0.9968
Epoch 00099: val_acc did not improve from 0.99679
248/248 [==============================] - 114s 458ms/step - loss: 3.2805e-04 - acc: 1.0000
Epoch 100/100
247/248 [============================>.] - ETA: 0s - loss: 9.9328e-04 - acc: 0.9997Epoch 1/1
107/248 [===========>..................] - ETA: 10s - loss: 0.0408 - acc: 0.9968
Epoch 00100: val_acc did not improve from 0.99679
248/248 [==============================] - 114s 460ms/step - loss: 9.9755e-04 - acc: 0.9997
```

Finally, once my model is trained and ready to go, performing inference on your model is very simplified with **ImageAI**. See the example below.

- **.predictMultipleImages()** , This function can be used to perform prediction on 2 or more im at once. Find example code, parameters of the function and returned values below

```
results_array = multiple_prediction.predictMultipleImages(all_images_array, result_count_per_i

for each_result in results_array:
    predictions, percentage_probabilities = each_result["predictions"], each_result["percentag
    for index in range(len(predictions)):
        print(predictions[index] , " : " , percentage_probabilities[index])
    print("----------------------")
```

https://imageai.readthedocs.io/en/latest/custom/

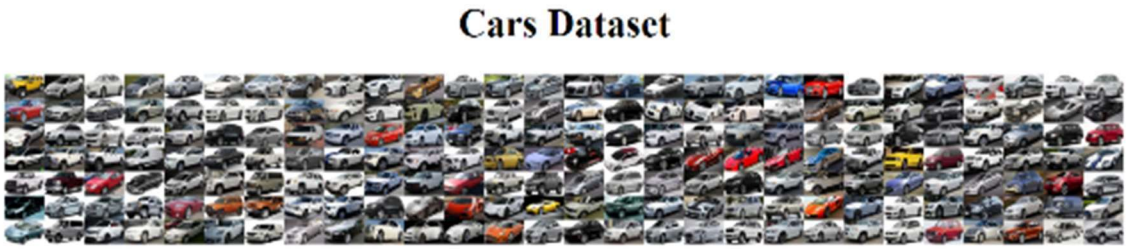
Building a Classification Dataset for my Prius C

My first objective was to identify any existing vehicle datasets out there which might be able to help accelerate my efforts. There are a handful of well-known vehicle datasets circulating around Kaggle. The most popular I've seen is the **Stanford Car Dataset**

featuring 196 classes. The "224" next to the zip file in the image below is meant to indicate they've been resized according to those dimensions for purposes your CNN. By far the largest dataset, however, is the **VMMRdb** Dataset.

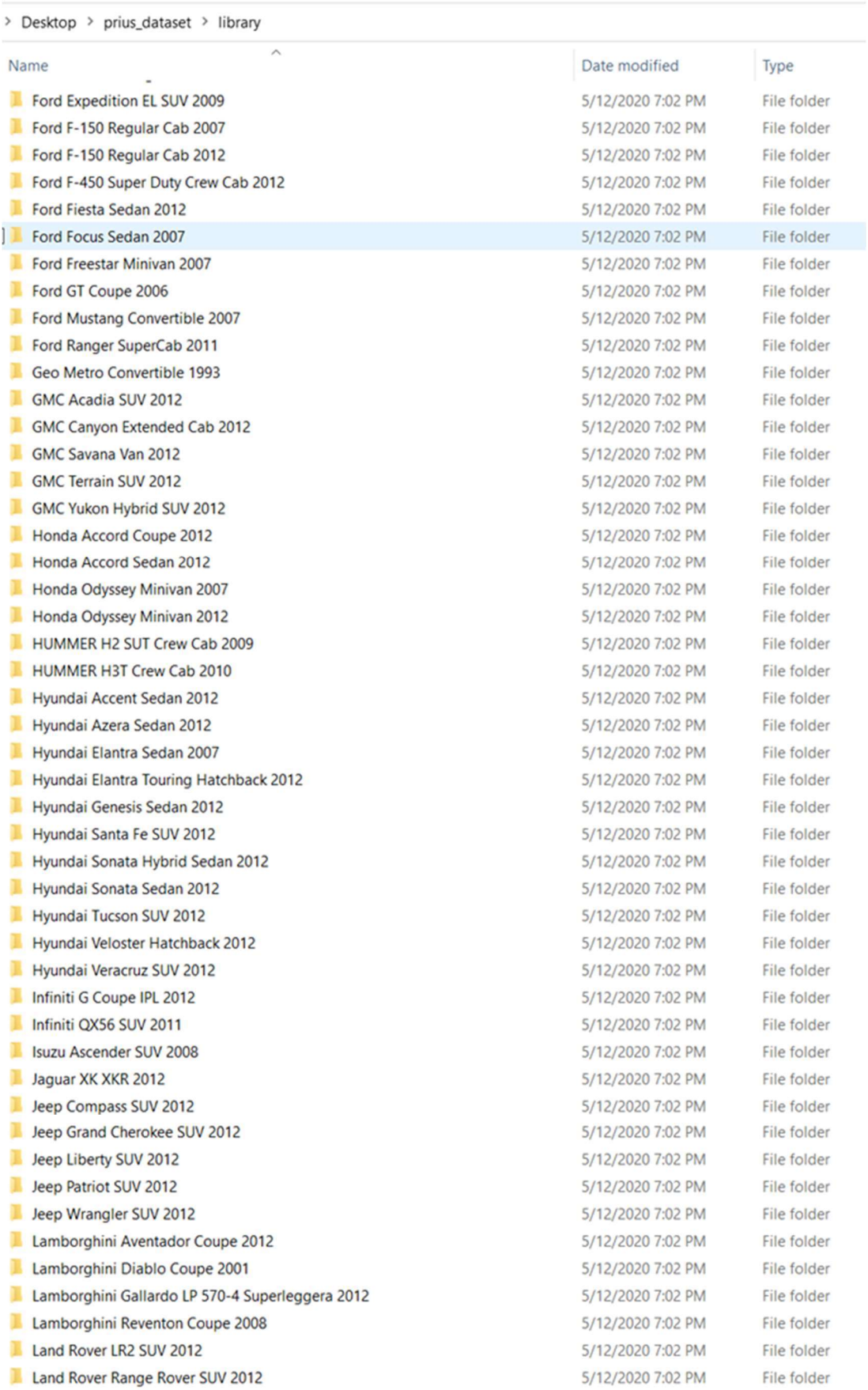
The Stanford Car Dataset was widely used on Kaggle but didn't contain any Prius C models. The VMMR dataset was massive. I used these to seed my "Vehicle" class. Because it was easy to have more "vehicle" images than "Prius C" images, I found if I had disproportionately more, it would cause my model to be biased. Nearly all the images were close-ups and perfect. I'm working with web cams at various resolutions so I knew I needed to likely acquire and label data from other sources in order to be very effective.

In total, the **Cars** dataset contains 16,185 images of 196 classes of cars. The data is split into 8,144 training images and 8,041 testing images, where each clas has been split roughly in a 50-50 split.



http://ai.stanford.edu/~jkrause/cars/car_dataset.html

The **Vehicle Make and Model Recognition dataset (VMMRdb)** contains 9,170 classes consisting of 291,752 images, covering models manufactured between 1950 and 2016. Here's an example of the resulting folder structure after unzipping the 13gb file:

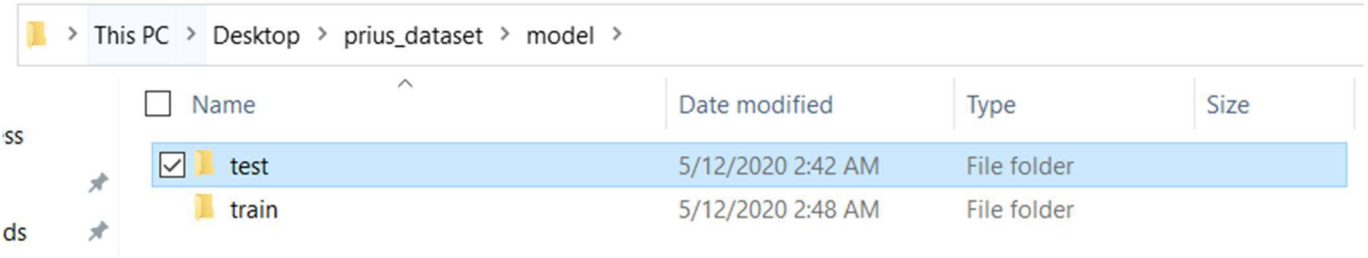


https://github.com/faezetta/VMMRdb
A Large and Diverse Dataset for Improved Vehicle Make and Model Recognition

Unfortunately, the **Cars** dataset did not include a Toyota Prius C. The **VMMRdb** did contain a small number of images for a 2012 Toyota Prius C but not nearly enough for training to be effective.

Folder Structure for Training Model

I used a test / train folder containing a Prius and Vehicle class each. There was a 70% / 30% split between images used in train versus test. I've seen recommendations vary between 50%/50% and 80%/20% split between train and test.



Building a Custom Dataset

There were a number of strategies here that I tested. At first, I was thinking I could just group all Prius models into one class irrespective of variation or year. This way, I have a larger sample size of data to use and Ill get some benefit from the roundabout way different models were similar. In terms of the non-matching class, I tried having just 1 class that was labeled "Vehicle" and included samples of many types of vehicles. I also tried including all the classes present in the datasets wondering if adding more classes provided more points for delineating differences and ultimately improve the effectiveness. I ultimately ended up going back through all my data and tried my best to clean out any example of a car which was not a Prius C for the Prius case. For the "Vehicles" case, I tried to provide a very rich pictures of vehicles that would not be a Prius C. This left me with just two classes.

There are also implications here in terms of your neural network structure and the relationship between your number of classes and the number of layers in your neural network. It seemed like most of this was abstracted away from you, however, when using **ImageAI.**

I had two more problems to solve:

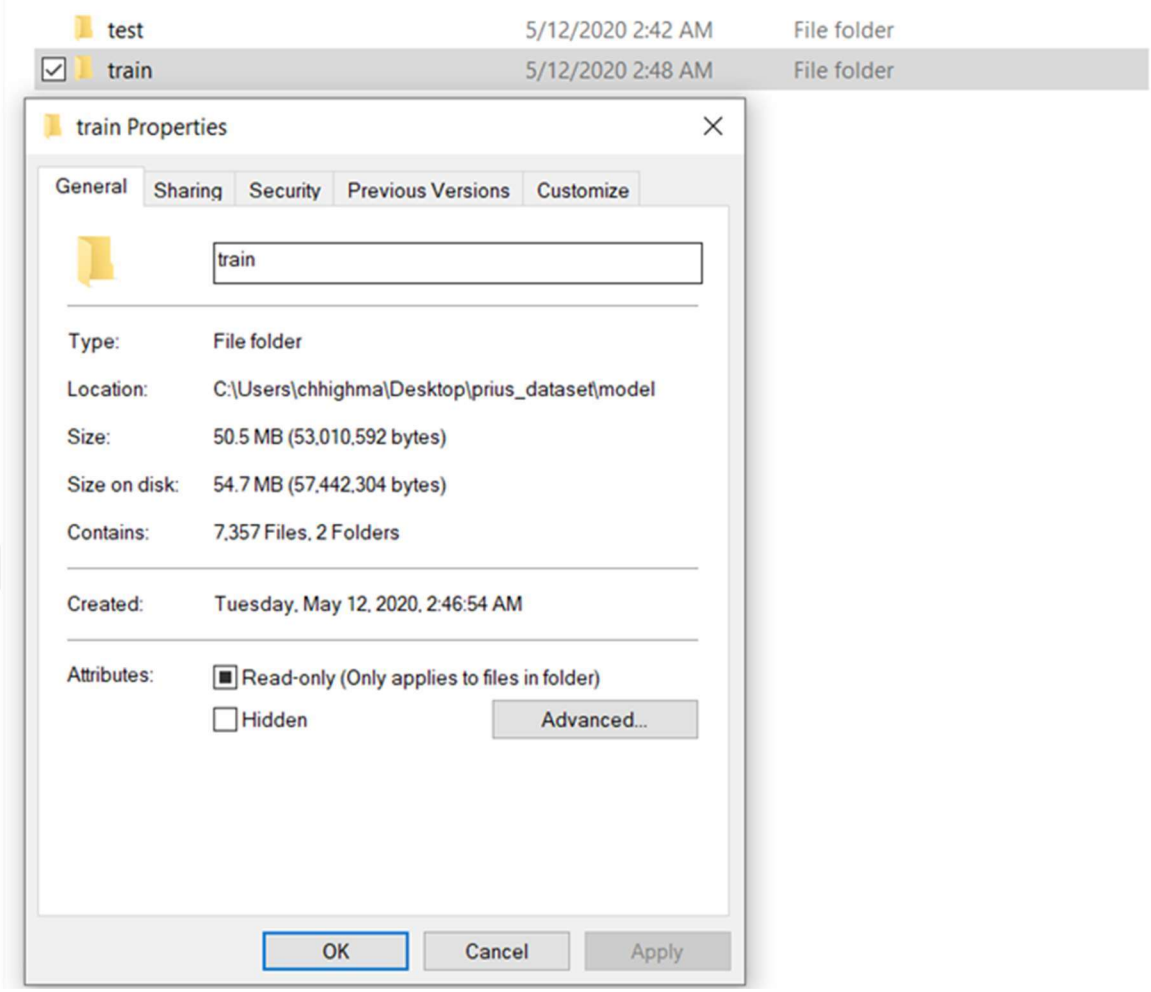
1.) I need to find many, many, many photos of Toyota Prius vehicles

2.) Ideally, these photos would be in more of a natural setting and not just contain vehicle close ups like you would expect from a commercial. Instead, because we'll be doing our inference from traffic cameras that will have odd angles on these vehicles to classify, it would be most ideal to find Prius C photos that are most aligned with the content we're hoping to predict.

I wrote a handful of scripts to do site scraping and experimented with various numbers of train/test data. Because my "Vehicle" folder was a generic folder really for any vehicle other than a Prius C, I became very curious if – for every vehicle type – I needed an example in both the train and test folders. I reasoned if the test folder did not have an example of a particular vehichle type which does exist in train, it could end up skewing results. I don't know this for certain but and I suppose you could argue not having a vehicle of the matching make/model could be an indicator of its effectiveness when classifying across a very complex object to learn.

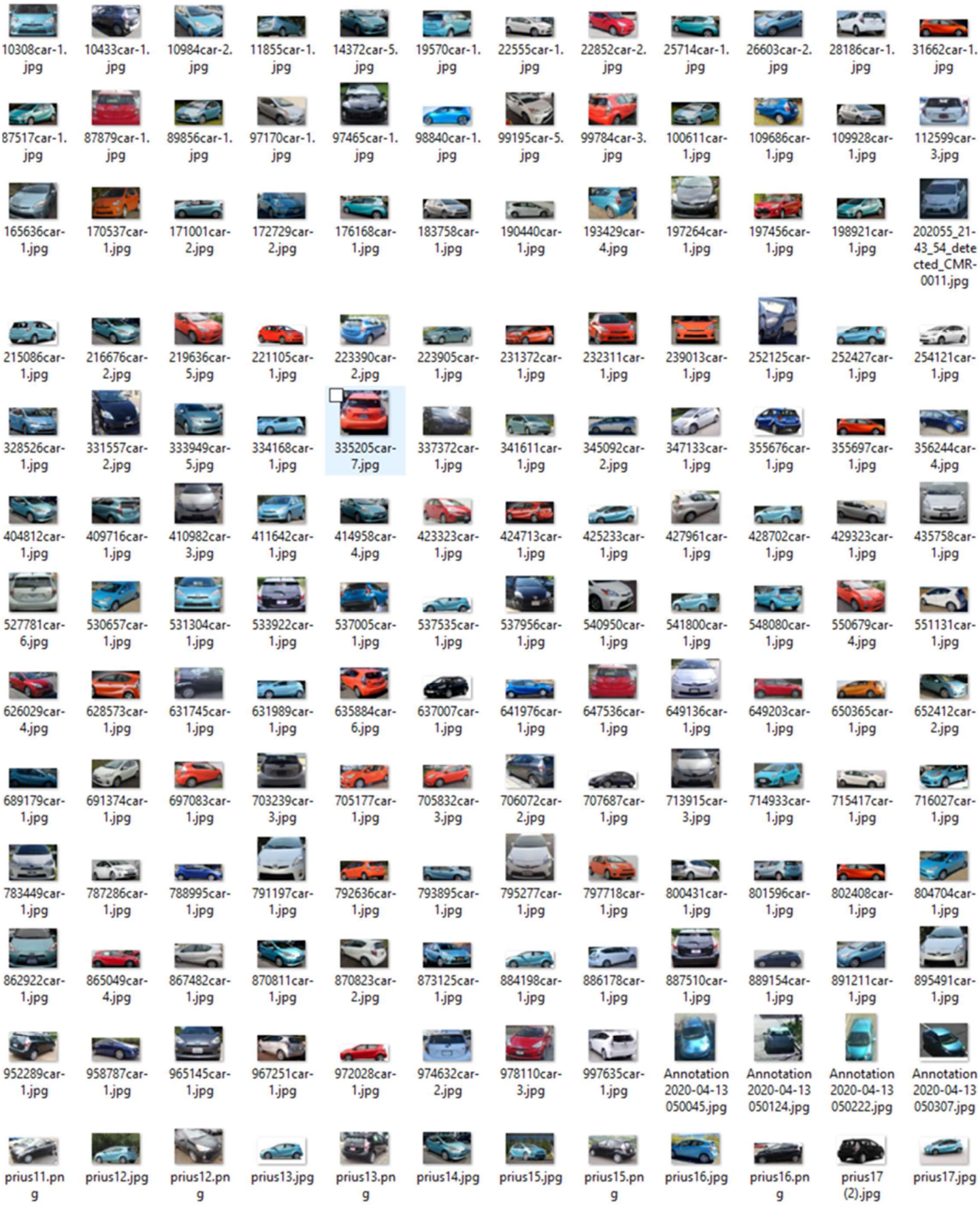
Final Dataset Size

Curating a clean, consistent customized dataset was almost certainly the most time-consuming piece of this project. I ended up with a total of 7,357 images across both my train and test folders.



Data Preparation

Preparing the data turned out to be quite trivial. I actually just ran all images through the first object detection / extraction step so that for every detected vehicle within an image, the contents within the bounded box are saved as their own image. This was a powerful normalization step because **ImageAI** will implictly end up resizing and scaling the images anyway. If half of the images scaled down and included other details in the photo, it would cause some type of bias in its ability to predict due to the inconsistency. Also, there is a parameter seen above called "*enhance_data*" which will go through all your training/test images and generate many **new images** as a result. For each image, a series of rotations, blurring, or other distortions would be applied. This adds more dimension and characteristics to the patterns it will be learning which should provide more variance to other artifacts or noise encountered with real images.

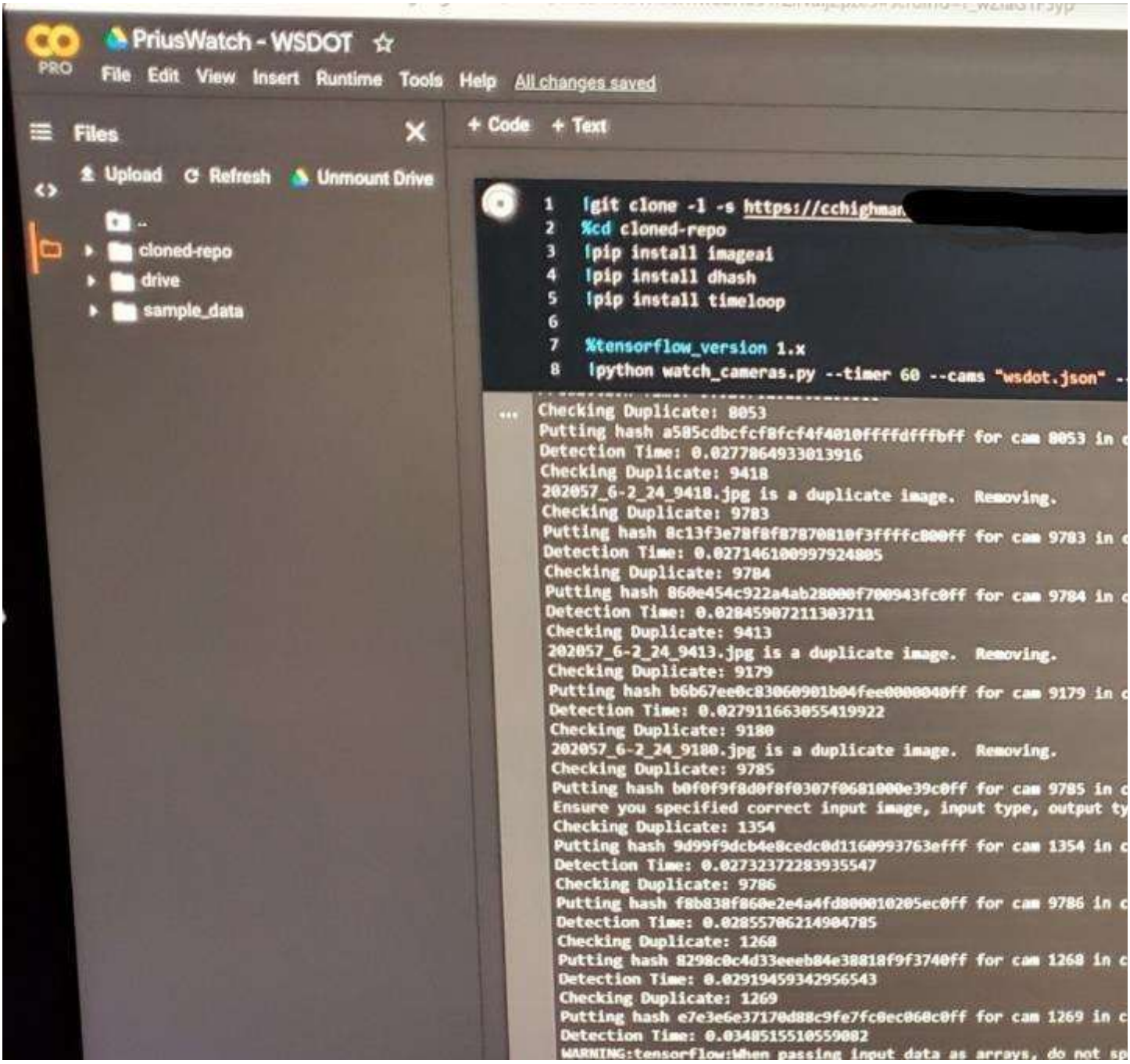

10308car-1.jpg
10433car-1.jpg
10984car-2.jpg
11855car-1.jpg
14372car-5.jpg
19570car-1.jpg
22555car-1.jpg
22852car-2.jpg
25714car-1.jpg
26603car-2.jpg
28186car-1.jpg
31662car-1.jpg
87517car-1.jpg
87879car-1.jpg
89856car-1.jpg
97170car-1.jpg
97465car-1.jpg
98840car-1.jpg
99195car-5.jpg
99784car-3.jpg
100611car-1.jpg
109686car-1.jpg
109928car-1.jpg
112599car-3.jpg
165636car-1.jpg
170537car-1.jpg
171001car-2.jpg
172729car-2.jpg
176168car-1.jpg
183758car-1.jpg
190440car-1.jpg
193429car-4.jpg
197264car-1.jpg
197456car-1.jpg
198921car-1.jpg
202055_21-43_54_detected_CMR-0011.jpg
215086car-1.jpg
216676car-2.jpg
219636car-5.jpg
221105car-1.jpg
223390car-2.jpg
223905car-1.jpg
231372car-1.jpg
232311car-1.jpg
239013car-1.jpg
252125car-1.jpg
252427car-1.jpg
254121car-1.jpg
328526car-1.jpg
331557car-2.jpg
333949car-5.jpg
334168car-1.jpg
335205car-7.jpg
337372car-1.jpg
341611car-1.jpg
345092car-2.jpg
347133car-1.jpg
355676car-1.jpg
355697car-1.jpg
356244car-4.jpg
404812car-1.jpg
409716car-1.jpg
410982car-3.jpg
411642car-1.jpg
414958car-4.jpg
423323car-1.jpg
424713car-1.jpg
425233car-1.jpg
427961car-1.jpg
428702car-1.jpg
429323car-1.jpg
435758car-1.jpg
527781car-6.jpg
530657car-1.jpg
531304car-1.jpg
533922car-1.jpg
537005car-1.jpg
537535car-1.jpg
537956car-1.jpg
540950car-1.jpg
541800car-1.jpg
548080car-1.jpg
550679car-4.jpg
551131car-1.jpg
626029car-4.jpg
628573car-1.jpg
631745car-1.jpg
631989car-1.jpg
635884car-6.jpg
637007car-1.jpg
641976car-1.jpg
647536car-1.jpg
649136car-1.jpg
649203car-1.jpg
650365car-1.jpg
652412car-2.jpg
689179car-1.jpg
691374car-1.jpg
697083car-1.jpg
703239car-3.jpg
705177car-1.jpg
705832car-3.jpg
706072car-2.jpg
707687car-1.jpg
713915car-3.jpg
714933car-1.jpg
715417car-1.jpg
716027car-1.jpg
783449car-1.jpg
787286car-1.jpg
788995car-1.jpg
791197car-1.jpg
792636car-1.jpg
793895car-1.jpg
795277car-1.jpg
797718car-1.jpg
800431car-1.jpg
801596car-1.jpg
802408car-1.jpg
804704car-1.jpg
862922car-1.jpg
865049car-4.jpg
867482car-1.jpg
870811car-1.jpg
870823car-2.jpg
873125car-1.jpg
884198car-1.jpg
886178car-1.jpg
887510car-1.jpg
889154car-1.jpg
891211car-1.jpg
895491car-1.jpg
952289car-1.jpg
958787car-1.jpg
965145car-1.jpg
967251car-1.jpg
972028car-1.jpg
974632car-2.jpg
978110car-3.jpg
997635car-1.jpg
Annotation 2020-04-13 050045.jpg
Annotation 2020-04-13 050124.jpg
Annotation 2020-04-13 050222.jpg
Annotation 2020-04-13 050307.jpg
prius11.png
prius12.jpg
prius12.png
prius13.jpg
prius13.png
prius14.jpg
prius15.jpg
prius15.png
prius16.jpg
prius16.png
prius17 (2).jpg
prius17.jpg

Inference in Practice

This notebook pulls down my private github repo, installs a handful of dependencies, and invokes a generously parameterized python script which carries out the work. The key parameters include a json document containing camera endpoints, an output location for matching images, accuracy considerations for inference, a name by which to use to identify results, and a model location. Google Drive was used as a the mapped location.

A basic hashmap is used to associate an image hash value to its camera id. This works out pretty well since it will not update or replace the hash value, or extraneously process a duplicate image, unless that the hashes no longer match.

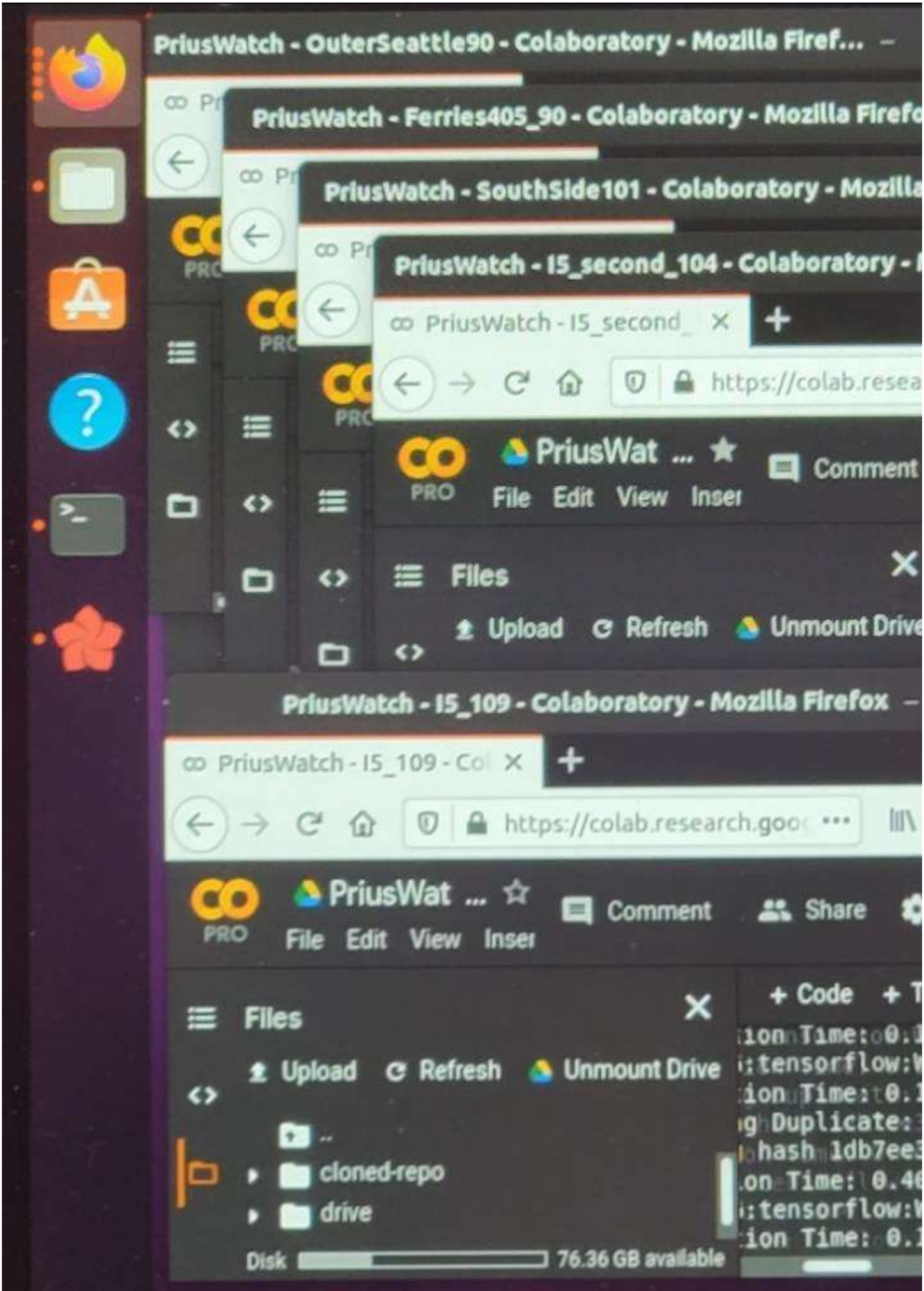
As we'll see next, this is just one of many copies of these notebooks that would run in parallel, dedicated each to whichever JSON they were parameterized with.



Starting to Scale Up

Each notebook is essentially doing the same but with some logical slice of the different camera endpoints as JSON. This approach was quite effective but did require more manual effort to perpetuate. In a perfect world where **Azure Databricks** was available with GPU resources, this work could easily be distributed and scaled far past the limitations present from using these multiple interfaces.

F


PriusWatch - OuterSeattle90 - Colaboratory - Mozilla Firef...
PriusWatch - Ferries405_90 - Colaboratory - Mozilla Firefo
PriusWatch - SouthSide101 - Colaboratory - Mozilla
PriusWatch - I5_second_104 - Colaboratory -
PriusWatch - I5_second_
https://colab.resea
PriusWat ...
Comment
File Edit View Inser
Files
Upload
Refresh
Unmount Drive
PriusWatch - I5_109 - Colaboratory - Mozilla Firefox
PriusWatch - I5_109 - Co
https://colab.research.goo
PriusWat ...
Comment
Share
File Edit View Inser
Files
Upload
Refresh
Unmount Drive
cloned-repo
drive
Disk
76.36 GB available
+ Code

Key Claims to Failure Fame

Throughout this write-up, I know I've mentioned more than once various strategies I've tried for different outcomes. Failure can look like a lot of things and express itself in ways that you really don't like!

For example…

1.) If your classifier isn't accurate, you'll quickly be accumulating "results" that havent really been processed yet
2.) Failure could be an infinite queue of pending writes to a path mapping back to Google Drive and all the matches you've found over the last 24 hours we're never recorded.
3.) Looking at the images below, and realizing you see all types of colors of cars being detected, but your intelligence isn't aware of color. This could be seen as a failure.
4.) So, you become resolute that you will make the code aware of color. Upon doing so, the bright lights of failure fame shine high when you realize how much impact lightning conditions have on color.

out_2020415_0-14_CMR-0255.jpg

out_2020415_0-14_CMR-0229.jpg

out_2020415_0-14_CMR-0170.jpg

out_2020415_0-14_CMR-0112.jpg

out_2020415_0-14_CMR-0109.jpg

out_2020415_0-14_CMR-0106.jpg

out_2020415_0-14_CMR-0038.jpg

out_2020415_0-14_CMR-0030.jpg

out_2020415_0-14_CMR-0020.jpg

out_2020415_0-10_CMR-0112.jpg

out_2020415_0-10_CMR-0109.jpg

out_2020415_0-10_CMR-0108.jpg

out_2020415_0-4_CMR-0230.jpg

out_2020415_0-4_CMR-0206.jpg

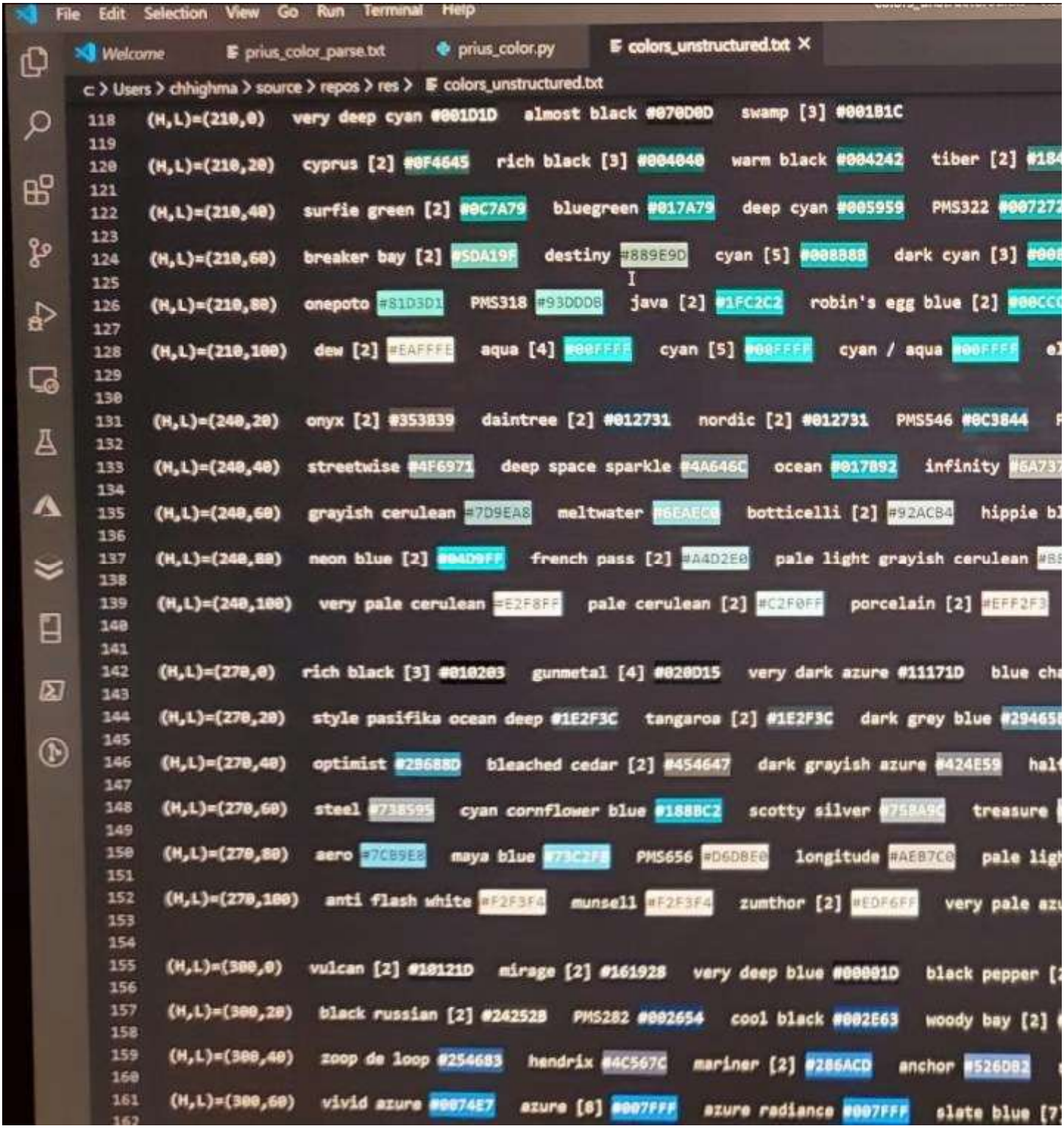
out_2020415_0-4_CMR-0203.jpg

A Bright, Colorful Idea….

A "Summer Rain" Toyota Prius has a rather unique color which uniquely distinguishes this vehicle from others. Would I even need a fancy classifier if I could simply detect the "Prius" color in an image? Sounds awesome, right? In practice, It wasn't that simple. I was able to find a massive text file of unstructured color labels and the associated hex value. VSCode shined very nicely having an extension which would show the color that maps to the hex value. There were thousands and thousands of colors. It was a massive text file. How the heck can I convert this into something usable? I also would discover, under various lightning conditions, the Prius may have shades that appear like any in the screenshot below. This could easily cause it match up with a gray or white vehicle. I discovered "Summer Rain" isnt one color, it's a gradient of colors. "Summer Rain" is more of a color palate than one particular color.

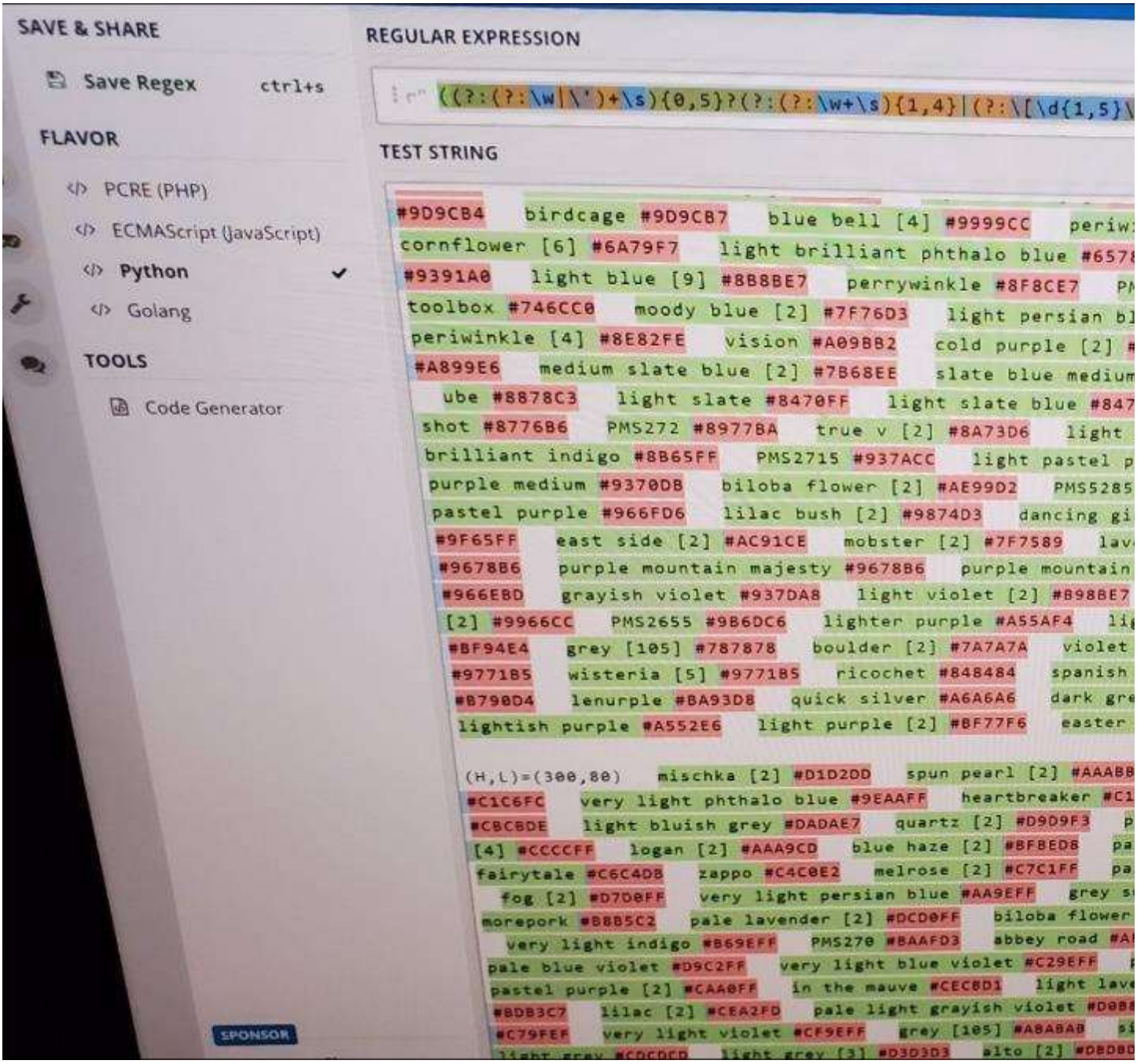
How exactly do you check the color of a vehicle? If I check the color of any given pixel, it could be any number of possible colors, especially over the entire image. Do I check the pixel in the center of the detected object dimensions? What if I end up checking the pixel in middle of the windshield? It's a bit more of a complicated problem. I ended up landing on a handful of strategies:

1.) **Number of Dominate Colors**
   In theory, the color of the vehicle should be one of the dominate colors in a pixel range representing a detected object as seen above.
2.) **Principal Component Analysis**
   This worked rather well but was a bit more costly.
3.) **Significant Color Contours**
   I went with method which was a part of OpenCV2 library.
4.) **Eucilidean Distance to Find Nearest Color**
   Once I returned back the three most significant color contours, I needed to see if the nearest color was a part of the "palate" I built for my Prius.



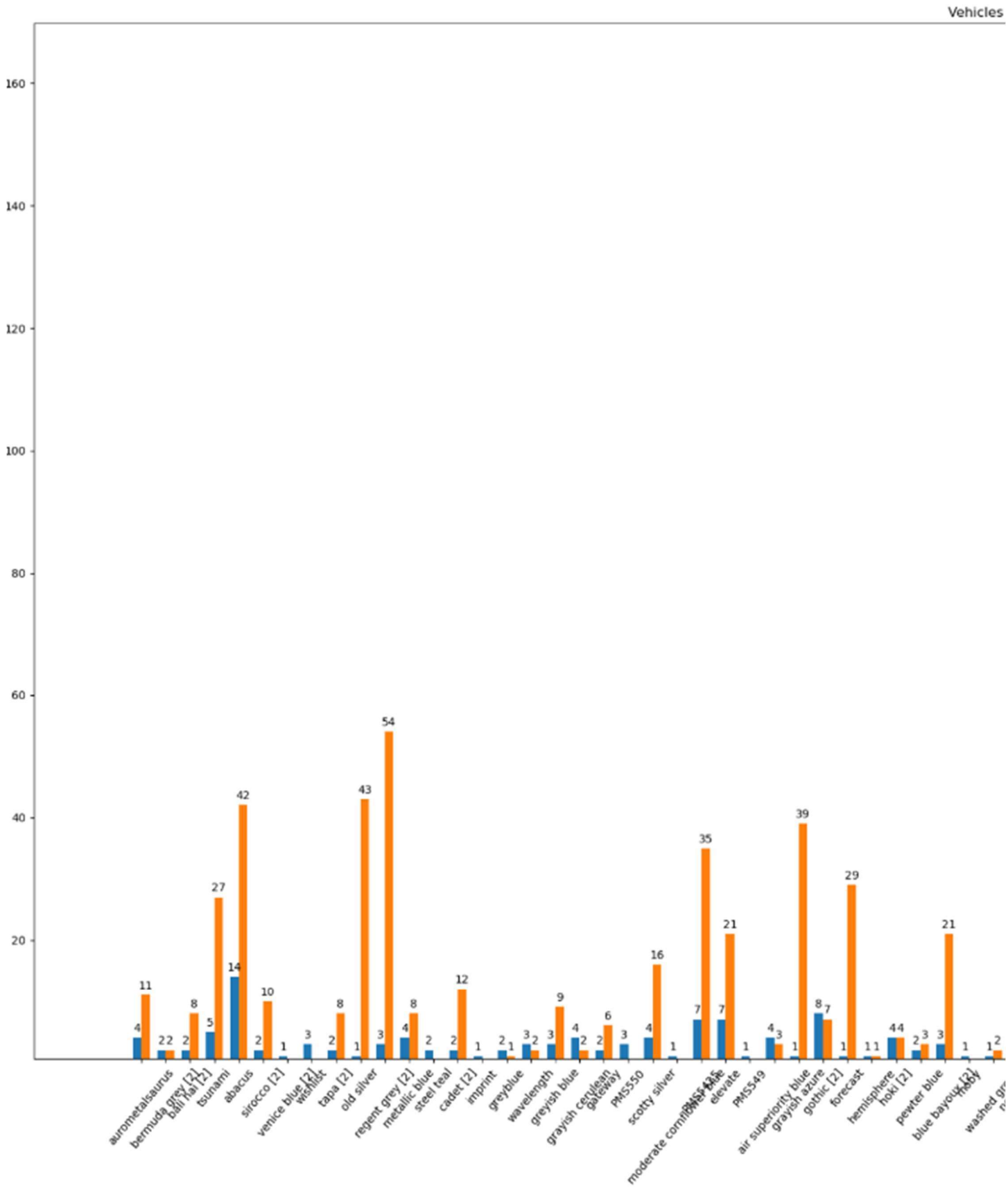
That moment when a regular expression changes everything

Ahh, when everything neatly extracts into two captures groups! This allowed for creation of a small Python script which effectively loads these key / value pairs into a hash map.



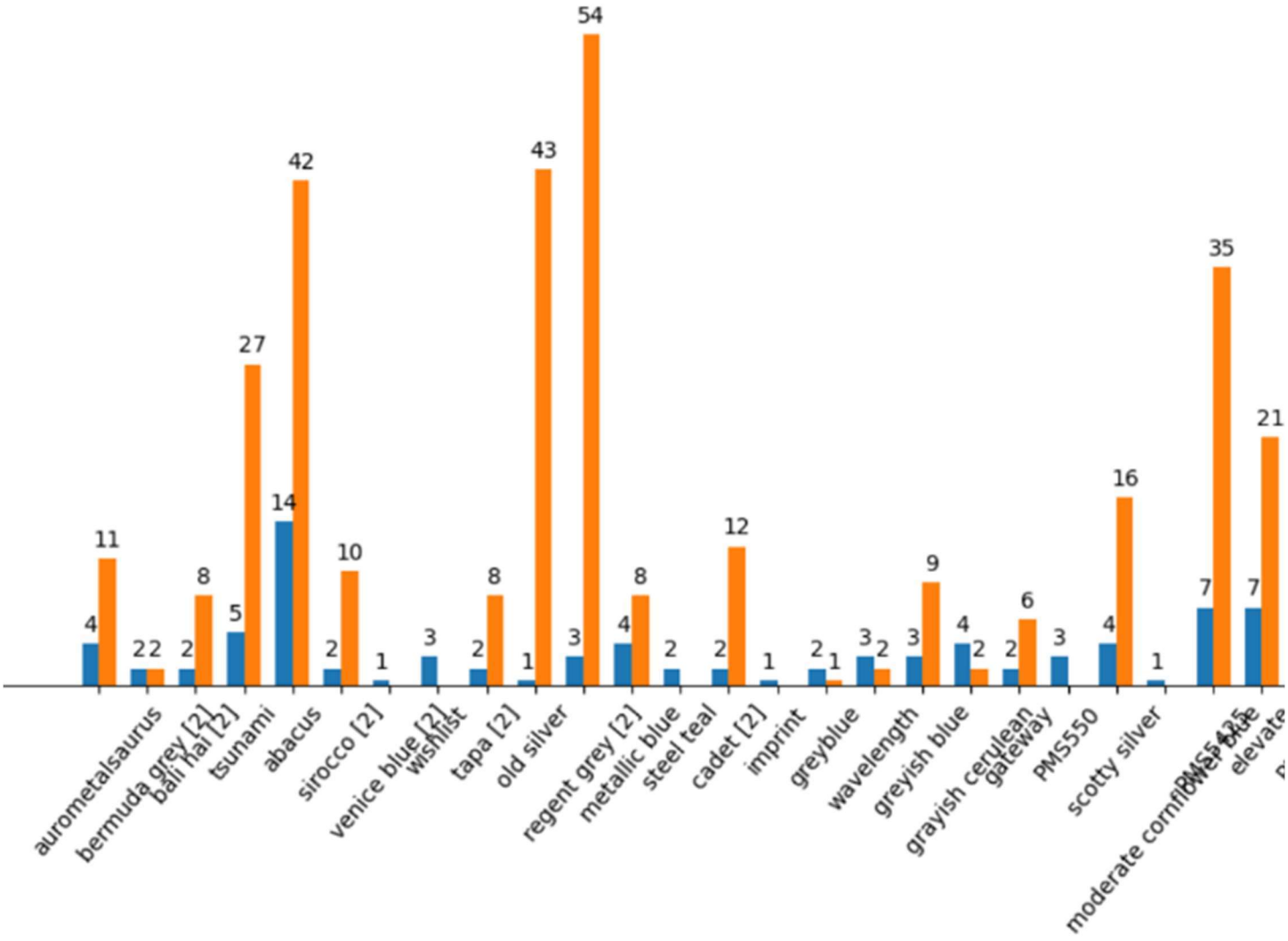
Vehicle Color Analytics – Zooming in

In order to understand what would really be the most effective color palate, I needed to perform some analytics. I analytics the most significant color contour across a large number of vehicles. The vehicles which were "Prius" were already labeled. When comparing the names of the detected colors below, I found often that a single Prius may have more than one significant color contour within the "palate" of a Prius based on the lighting conditions. This discovery caused me to adjust my logic to ask if at least 2 of 3 of the most significant color contours had a nearest color within the Prius palate color range.


Vehicles
160
140
120
100
80
60
40
20
4 11
2 2
2 8
5 27
14 42
2 10
1
3
2 8
1 43
3 54
4 8
2
2 12
1
2 1
3 2
3 9
4 2
2 6
3
4 16
1
7 35
7 21
1
4 3
1 39
8 7
1 29
1 1
4 4
2 3
3 21
1
1 2
tsunami
abacus
sirocco [2]
tapa [2]
old silver
regent grey [2]
metallic blue
steel teal
cadet [2]
imprint
greyblue
wavelength
greyish blue
grayish cerulean
gateway
PM5550
scotty silver
elevate
PM5549
air superiority blue
grayish azure
gothic [2]
forecast
hemisphere
hoki [2]
pewter blue
blue bayoux [2]

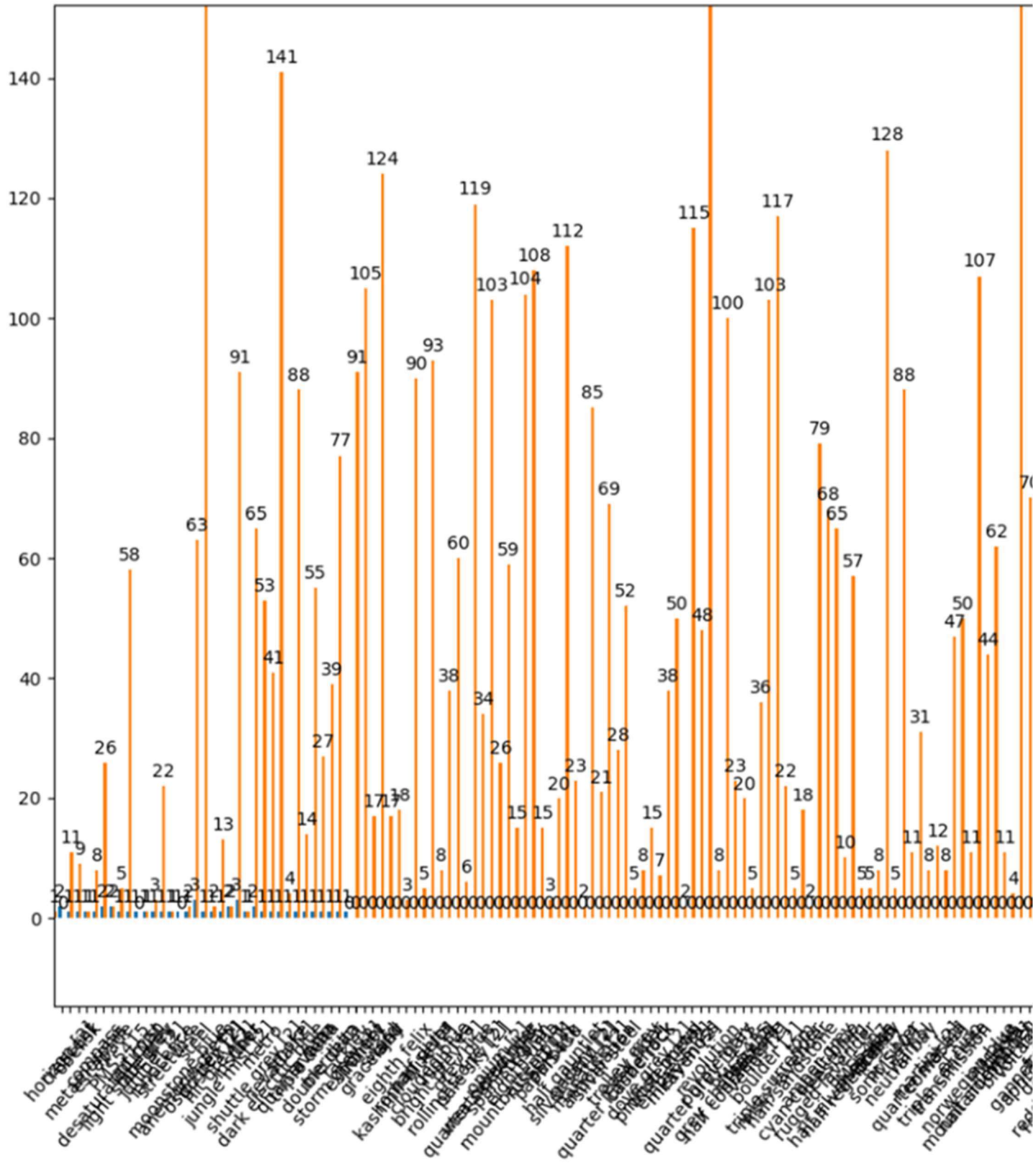
Vehicle Color Analytics – Even Closer

I found most of the variation in colors between the detects contours for the Prius were due to lightning conditions.



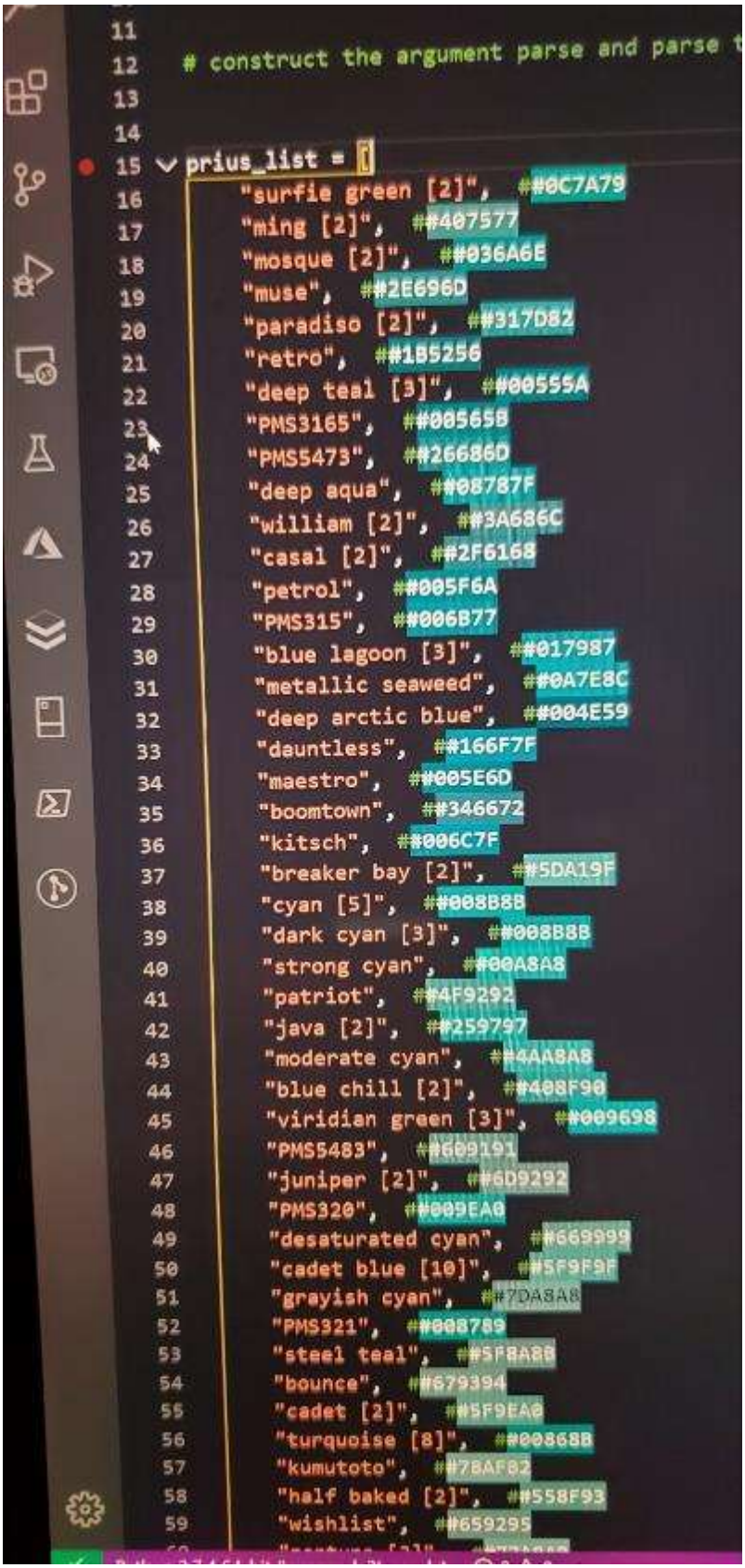
Full Color Analysis – "Summer Rain" vs Vehicles

Looking at the full picture, you can barely see the blue tick marks for the Prius vehicles on the bottom left. The key takeaways from this analysis was that the Prius color palate range was indeed quite unique and should be a valuable factor in identifying a "Summer Rain" Prius based on color alone. There were caveats. Depending on lighting conditions, you could end up with a false positive for many other popular vehicle types. Using the strategy of checking to see if multiple significant contours existed in the Prius palate did help.



Summer Rain Color Palate

Below is a screenshot of what the “Prius” color plate ended up looking like, at least partially.



```
11
12  # construct the argument parse and parse t
13
14
15  prius_list = [
16      "surfie green [2]",  ##0C7A79
17      "ming [2]",  ##407577
18      "mosque [2]",  ##036A6E
19      "muse",  ##2E696D
20      "paradiso [2]",  ##317D82
21      "retro",  ##1B5256
22      "deep teal [3]",  ##00555A
23      "PMS3165",  ##00565B
24      "PMS5473",  ##26686D
25      "deep aqua",  ##08787F
26      "william [2]",  ##3A686C
27      "casal [2]",  ##2F6168
28      "petrol",  ##005F6A
29      "PMS315",  ##006B77
30      "blue lagoon [3]",  ##017987
31      "metallic seaweed",  ##0A7E8C
32      "deep arctic blue",  ##004E59
33      "dauntless",  ##166F7F
34      "maestro",  ##005E6D
35      "boomtown",  ##346672
36      "kitsch",  ##006C7F
37      "breaker bay [2]",  ##5DA19F
38      "cyan [5]",  ##008B8B
39      "dark cyan [3]",  ##008B8B
40      "strong cyan",  ##00A8A8
41      "patriot",  ##4F9292
42      "java [2]",  ##259797
43      "moderate cyan",  ##4AA8A8
44      "blue chill [2]",  ##408F90
45      "viridian green [3]",  ##009698
46      "PMS5483",  ##609191
47      "juniper [2]",  ##6D9292
48      "PMS320",  ##009EA0
49      "desaturated cyan",  ##669999
50      "cadet blue [10]",  ##5F9F9F
51      "grayish cyan",  ##7DA8A8
52      "PMS321",  ##008789
53      "steel teal",  ##5F8A8B
54      "bounce",  ##679394
55      "cadet [2]",  ##5F9EA0
56      "turquoise [8]",  ##00868B
57      "kumutoto",  ##78AF82
58      "half baked [2]",  ##558F93
59      "wishlist",  ##659295
```

Significant Color Contour Logic

Screenshot showing some of the logic used to find significant color contours.

Instrumental in navigating the world of pixels, shades, and contours.
https://www.pyimagesearch.com/

```
    with open(filename, "w") as f:
        json.dump(data, f, indent=4)


def find_significant_contour(img):
    contours, hierarchy = cv2.findContours(
        img,
        cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE
    )

    # Find level 1 contours
    level1Meta = []
    for contourIndex, tupl in enumerate(hierarchy[0]):
        # Each array is in format (Next, Prev, First child,
        # Filter the ones without parent
        if tupl[3] == -1:
            tupl = np.insert(tupl.copy(), 0, [contourIndex]
            level1Meta.append(tupl)

    # From among them, find the contours with large surface
    contoursWithArea = []
    for tupl in level1Meta:
        contourIndex = tupl[0]
        contour = contours[contourIndex]
        area = cv2.contourArea(contour)
        contoursWithArea.append([contour, area, contourIndex,

    contoursWithArea.sort(key=lambda meta: meta[1], reverse=T
    topThree = []
    for i in range(0, min(len(contoursWithArea) - 1, 2)):
        topThree.append(contoursWithArea[i][0])

    return topThree


def get_contour_colors(image_src):
    # load the image and resize it to a smaller factor so that
    # the shapes can be approximated better
    # image = cv2.imread(os.path.join(path,image))
    return get_contour_colors_from_array(cv2.imread(image_src))


def get_contour_colors_from_array(image):
    # load the image and resize it to a smaller
```

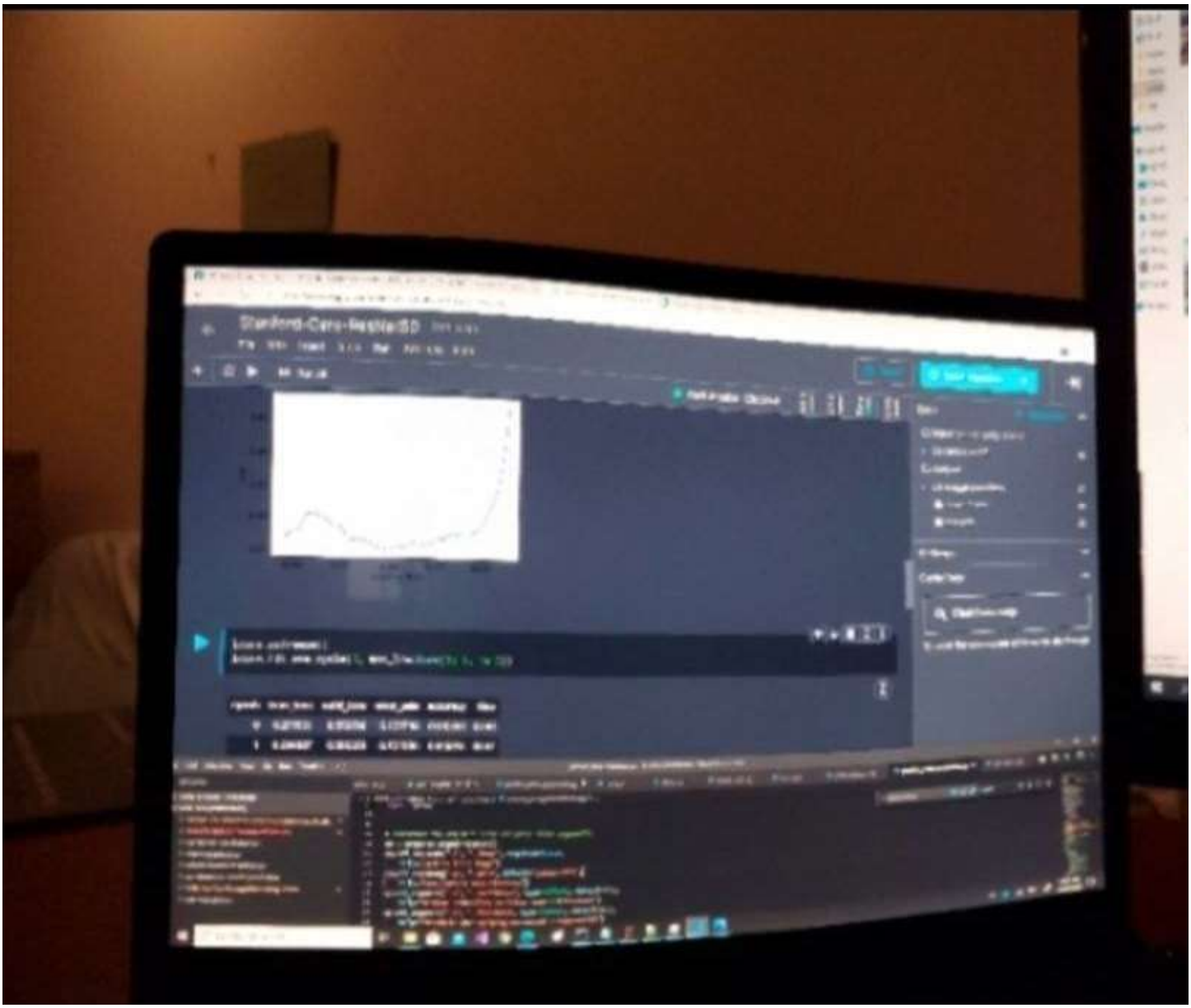
Labeling Vehicle Colors

Screenshot of logic used to label vehicles with a respective color contour.

```python
def detect_color(image_src):
    return detect_color_from_array(cv2.imread(image_src)


def detect_color_from_array(image):
    # load the image and resize it to a smaller factor so
    # the shapes can be approximated better
    # image = cv2.imread(os.path.join(path,image))
    try:
        resized = imutils.resize(image, width=300)
        ratio = image.shape[0] / float(resized.shape[0])
        # blur the resized image slightly, then convert it
        # grayscale and the L*a*b* color spaces
        blurred = cv2.GaussianBlur(resized, (5, 5), 0)
        gray = cv2.cvtColor(blurred, cv2.COLOR_BGR2GRAY)
        lab = cv2.cvtColor(blurred, cv2.COLOR_BGR2LAB)
        thresh = cv2.threshold(gray, 60, 255, cv2.THRESH_B

        # find contours in the thresholded image
        cnts = find_significant_contour(thresh.copy())
        cl = ColorLabeler()

        contours = []

        for c in cnts:
            color = cl.label(lab, c)
            print("Color: " + color)
            contours.append(color)

        return contours
    except Exception as e:
        print("While detecting color: " + str(e))


def has_prius_color(image):
    detected_color = detect_color(image)
    if detected_color in prius_colors:
        return True
    return False


def has_prius_color_from_array(image):
    # Top 3 contours
    detected_colors = detect_color_from_array(image)
    for color in detected_colors:
        if color in prius_colors:
            return color
    return None
```

BINGO! CAPTURED MY CAR ON CAMERA

I nearly fell out of my seat. Ironically enough, it was not captured on any other camera downtown that I could discern. In the next picture, notice the missing rear hubcab and the missing front license plate. Yep, that's my lovely Prius! First owner, driven coast to coast multiple times! At this point, I was still keeping copies of every snapshot, so you can imagine I went back and carefully combed lots and lots of images looking for cases that it wasn't picked up or maybe could be seen in the distance. My entire weekend was spent staring this way but at the end, not a single trace. It fueled my desire to upgrade my abilities to support live stream cameras.



Nice catch huh ???

This was one heck of a great match. I can only credit the ability to match this fuzziness as most likely due to the implicit **image augmentation** done to further enhance datasets that create blurred/distorted copies of existing images. I was hoping to follow it's trail across cameras and find a roundabout area where it was last seen, ideally spot the vehicle, and call the cops. Then, when they swoop in, I'd have one heck of a sweet video I could create arriving on the scene to recover my vehicle!!!! I **did** get my car back but it went down just a little bit differently….

Unable to scale even with Tesla P100 GPU

In the above case with my vehicle, I was on a 12 hour delay because of the amount of time it took for compute to churn through all the images. As seen below, I needed inference to move much faster than this. It finally dawned on me that my bottle neck was likely the fact I was writing every image to disk. By this point, I had already ran out of space several times and even trying to delete the images was time consuming. As you may had noticed in the above code screenshots, there are *"_from_array"* suffixed methods which just pass around in-memory numpy arrays. After I re-ran plumbing to no longer save to disk or read from disk, the power of a GPU began to shine.

```
2020-05-01 01:05:51.665026: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
task:0/device:GPU:0 with 15216 MB memory) -> physical GPU (device: 0, name: Tesl
ility: 6.0)
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/python/ops/
le.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint i
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/python/ops/
y_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Single Thread - Processor Count: 2
Populating images
Detecting vehicle for 2020430_17-21_29_redmond-20.jpg -> ../drive/My Drive/new_r
2020-05-01 01:07:42.428137: E tensorflow/core/grappler/optimizers/meta_optimizer
have computed start >= end since stride is negative, but is 0 and 2 (computed fr
nk 2 and stride-1)
2020-05-01 01:07:42.629459: E tensorflow/core/grappler/optimizers/meta_optimizer
ave computed start >= end since stride is negative, but is 0 and 2 (computed frc
k 2 and stride-1)
2020-05-01 01:07:42.871163: E tensorflow/core/grappler/optimizers/meta_optimizer
have computed start >= end since stride is negative, but is 0 and 2 (computed fr
nk 2 and stride-1)
2020-05-01 01:07:43.624771: I tensorflow/stream_executor/platform/default/dso_lc
so.7
2020-05-01 01:07:47.985630: I tensorflow/stream_executor/platform/default/dso_lc
s.so.10
Prediction Time: 175.67332220077515
Detecting vehicle for 2020430_17-21_29_redmond-3.jpg -> ../drive/My Drive/new_re
WARNING:tensorflow:When passing input data as arrays, do not specify `steps_per_
Prediction Time: 3.4103124141693115
Detecting vehicle for 2020430_17-21_29_redmond-16.jpg -> ../drive/My Drive/new_r
WARNING:tensorflow:When passing input data as arrays, do not specify `steps_per_
Prediction Time: 1.0843513011932373
Detecting vehicle for 2020430_17-21_29_redmond-12.jpg -> ../drive/My Drive/new_r
Prediction Time: 0.9256045818328857
Detecting vehicle for 2020430_17-21_29_redmond-15.jpg -> ../drive/My Drive/new_r
Prediction Time: 1.2569677829742432
Detecting vehicle for 2020430_17-21_29_redmond-21.jpg -> ../drive/My Drive/new_r
Prediction Time: 0.8175146579742432
Detecting vehicle for 2020430_17-21_29_redmond-8.jpg -> ../drive/My Drive/new_re
WARNING:tensorflow:When passing input data as arrays, do not specify `steps_per_
Prediction Time: 1.2308285236358643
Detecting vehicle for 2020430_17-21_29_redmond-30.jpg -> ../drive/My Drive/new_r
Prediction Time: 1.9244060516357422
Detecting vehicle for 2020430_17-21_29_redmond-22.jpg -> ../drive/My Drive/new_r
WARNING:tensorflow:When passing input data as arrays, do not specify `steps_per_
Prediction Time: 1.2179932594299316
Detecting vehicle for 2020430_17-21_29_redmond-13.jpg -> ../drive/My Drive/new_r
WARNING:tensorflow:When passing input data as arrays, do not specify `steps_per_
Prediction Time: 2.0579633712768555
Detecting vehicle for 2020430_17-21_29_redmond-2.jpg -> ../drive/My Drive/new_re
```

To infinity…and beyond!

Going from 1-3 seconds to **~0.012 seconds** made a massive difference!!! I was running through predictions so quickly that all I could do was think aout how many other cities / counties in the State of Washington had cameras. Turns out, many did. I began adding every where from Spokane, Olympia, Auburn, Everett, and many others until my total camera count was ~4,000. By this time, we were in the second week of May when I began to eye processing live video feeds as the next extension. There are plenty of them out there.

PriusWatch - Seattle200.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on May 5

+ Code + Text

```
[ ]  1  %tensorflow_version 1.x
     2  !python watch_cameras.py --timer 60 --cams "seattle200.json" --ou
```

```
WARNING:tensorflow:When passing input data as arrays, do not specify
Prediction Time: 0.0181581974029541
WARNING:tensorflow:When passing input data as arrays, do not specify
Prediction Time: 0.019381999969482422
Checking Duplicate: CMR-0259
Putting hash 969e1d59891e8ece030efcf1a3333f8d for cam CMR-0259 in cach
Detection Time: 0.056203365325927734
WARNING:tensorflow:When passing input data as arrays, do not specify
Prediction Time: 0.0234067440032959
Checking Duplicate: CMR-0202
Putting hash e8680a08213330300140a01818cf0038 for cam CMR-0202 in cach
Detection Time: 0.06189417839050293
WARNING:tensorflow:When passing input data as arrays, do not specify
Prediction Time: 0.019653797149658203
WARNING:tensorflow:When passing input data as arrays, do not specify
Prediction Time: 0.018268823623657227
WARNING:tensorflow:When passing input data as arrays, do not specify
Prediction Time: 0.01865363121032715
WARNING:tensorflow:When passing input data as arrays, do not specify
Prediction Time: 0.01796126365661621
WARNING:tensorflow:When passing input data as arrays, do not specify
Prediction Time: 0.017473697662353516
WARNING:tensorflow:When passing input data as arrays, do not specify
Prediction Time: 0.017684221267700195
WARNING:tensorflow:When passing input data as arrays, do not specify
Prediction Time: 0.017876148223876953
WARNING:tensorflow:When passing input data as arrays, do not specify
Prediction Time: 0.02051568031311035
WARNING:tensorflow:When passing input data as arrays, do not specify
Prediction Time: 0.028804779052734375
WARNING:tensorflow:When passing input data as arrays, do not specify
Prediction Time: 0.01758861541748047
Checking Duplicate: CMR-0155
Putting hash 381932737afcf8d3009f77763f7fe1ff for cam CMR-0155 in cach
Detection Time: 0.038115739822387695
WARNING:tensorflow:When passing input data as arrays, do not specify
Prediction Time: 0.019172906875610351
Checking Duplicate: CMR-0194
Putting hash d0b1b992b360cca402c40d9fb763cb9d for cam CMR-0194 in cach
Detection Time: 0.0468902587890625
Checking Duplicate: CMR-0188
Putting hash 391a2a71993cc2b901303ec6fb790413 for cam CMR-0188 in cach
Detection Time: 0.05648970603942871
WARNING:tensorflow:When passing input data as arrays, do not specify
```

Accurate, real-time vehicle classification

99.99710321426392-99.99710321426392-99.99710321426392-99.99710321426392-out_images.jpg

2020412_14-31_bellevue-075n.jpg

2020412_14-32_962

out_2020412_14-20_9462.jpg

out_2020412_14-32_9626.jpg

out_2020412_14-35_redm

out_2020412_15-29_9462.jpg

out_2020412_15-54_9458.jpg

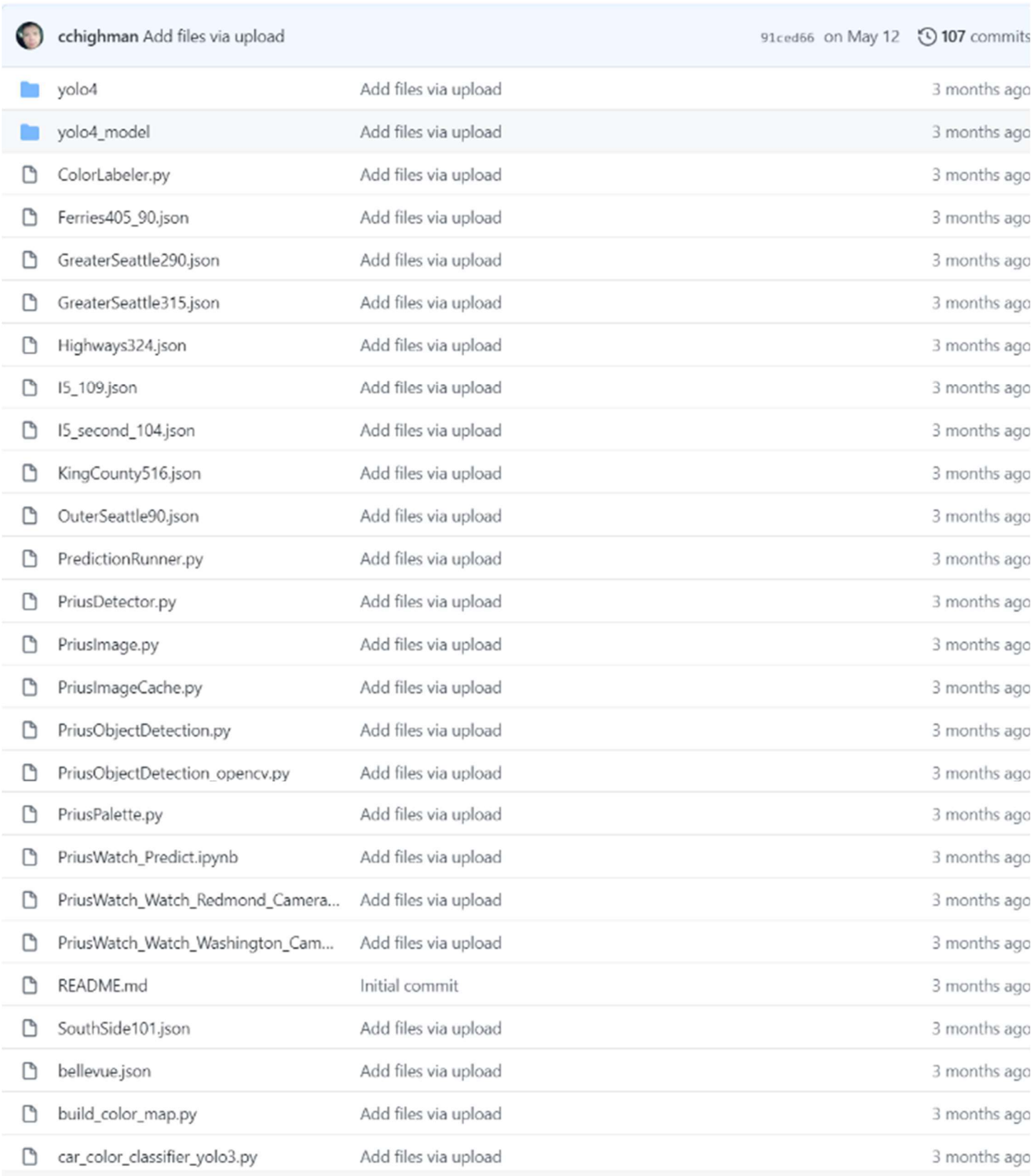
out_2020412_15-56_CMR

perfect_2020419_18-4_104.jpg

perfect_2020419_18-7_9495.jpg

perfect_2020420_16-16_

Project currently in a private Github repo



cchighman Add files via upload	91ced66 on May 12	**107** commits
yolo4	Add files via upload	3 months ago
yolo4_model	Add files via upload	3 months ago
ColorLabeler.py	Add files via upload	3 months ago
Ferries405_90.json	Add files via upload	3 months ago
GreaterSeattle290.json	Add files via upload	3 months ago
GreaterSeattle315.json	Add files via upload	3 months ago
Highways324.json	Add files via upload	3 months ago
I5_109.json	Add files via upload	3 months ago
I5_second_104.json	Add files via upload	3 months ago
KingCounty516.json	Add files via upload	3 months ago
OuterSeattle90.json	Add files via upload	3 months ago
PredictionRunner.py	Add files via upload	3 months ago
PriusDetector.py	Add files via upload	3 months ago
PriusImage.py	Add files via upload	3 months ago
PriusImageCache.py	Add files via upload	3 months ago
PriusObjectDetection.py	Add files via upload	3 months ago
PriusObjectDetection_opencv.py	Add files via upload	3 months ago
PriusPalette.py	Add files via upload	3 months ago
PriusWatch_Predict.ipynb	Add files via upload	3 months ago
PriusWatch_Watch_Redmond_Camera...	Add files via upload	3 months ago
PriusWatch_Watch_Washington_Cam...	Add files via upload	3 months ago
README.md	Initial commit	3 months ago
SouthSide101.json	Add files via upload	3 months ago
bellevue.json	Add files via upload	3 months ago
build_color_map.py	Add files via upload	3 months ago
car_color_classifier_yolo3.py	Add files via upload	3 months ago

..

And then the phone rang…

Redmond Police had recovered my stolen Prius. It was parked across the street from a bar in northern Seattle and someone called it in as a suspicious vehicle. It wreaked of alcohol and had several blunts rolled up on the dashboard. I picked it up from a tow yard of N Aurora Ave. Other than that, it had absolutely no damage. I was very lucky.

To help illustrate how I felt, just like that, the countless hours I placed into this on nights and weekends for 5 weeks – much like the incredible length of this email and its final conclusion – was finished.

Nothing else, nothing more. Just a rediculously refined amount of effort sitting in a repo.

But, I do have my car, I did at least see it once through means of the tech, it was an awesome learning experience, and maybe I can pass it along somehow for a good person if ethical/good AI guidelines can somehow be ensured.

Anyone have any use for any of this research ??

Thanks for reading!

Christopher