Ethan Taylor Behar
Dr. Wang
B546: Project Final Report - Testing Tesla's Vision
10/21/2020

1 Introduction

My motivation for this project can be summed up in a few words. Essentially because I have watched Tesla's object detection learn from its release, I am curious if I could possibly trick the object detection.

Additionally, because Tesla's Full Self-Driving (FSD) capabilities is now released it is paramount to ensure the system is resistant to attacks. At the time of my proposal, roughly two months ago, Tesla's FSD capabilities were not released. Now the beta release has begun for FSD; the first release was on October 21st. Tesla is preparing for its 5th wave release of the FSD software. With every release more users are invited to the beta to test the software. But beta testing FSD is not nearly the same as beta testing a videogame or operating system. These users are risking their lives when testing this software. If the FSD software has a bug or vulnerability people can get hurt. It is for this reason I attempted to test Tesla's vision and aimed to prove it is strong and robust.

My initial project had four distinct milestones:

- Build an object detection model that can recognize bicycles.
- Research adversarial examples (AE) and apply one to my dataset. Test how my model responds to the AE.
- Take the original and attacked dataset and see if Tesla can visualize these datasets.
- Draw a hypothesis on the discrepancies between both object detection systems.

I have successfully completed the first two tasks. I used Detectron2 [1] as my machine learning system and implemented the Dense Adversary Generation algorithm presented by Cihang et al [2]. Although, there was a need for a slight modification with my plans for my object detection model. Originally, I had only planned to detect bicycles, but I needed to include another object. Section 3.2 expands on why there was a need to have other objects. The third task was a complete failure. In short, my Tesla Model 3 did not visualize any of the pictures I placed near it. Due to this failure my fourth tasks could not be completed. So, I modified my fourth task to draw a conclusion as to why my Model 3 did not visualize any of the pictures. Section 4.3 discusses this conclusion. Additionally, I draw a conclusion on an interesting fact that I noticed when evaluating the results of my AE. Section 4.2 discusses this conclusion.

2 Related Work

There is a lot of research regarding machine learning and object detection. In this section I only discuss some of the most relevant related works.

The first related works I will discuss is what my project should have explored. But due to limitations with my Model 3's hardware I could not pursue this avenue of research. There are three published research papers: Kevin et al [3], Ivan et al [4], and Yue et al [5] that build physical adversarial examples (AE) that can be applied to physical objects. All three papers attacked stop signs and other physical objects. The papers were successful in tricking various objecting detection systems into misclassifying stop signs. The danger of this is obvious and apparent. This is why research along these lines is paramount. Companies must ensure their automated driving systems (ADS) are safe, secure, and resistant enough to avoid being tricked. Along with stop signs, other objects are attack but are not relevant to this project or ADSs so I do not mention them.

The next two related works explore AEs and applying them to images. Goodfellow et al [6] developed the fast gradient sign method (FGSM) AE. This paper focuses on neural networks' linear nature. Neural networks are linearized so they can be easily optimized.

FGSM uses this knowledge to its advantage to exploit linear neural networks. Thus, proving linear neural networks are more susceptible to AEs. Alexey et al [7] takes three methods of generating AEs and proves all three can be used on images of physical objects taken from a cell-phone camera. The three AEs they used are: FGSM, an iterative extension of FGSM, and the iterative least-likely category method. This last method was developed to work on models with a much larger number of categories. Whereas the first two work well on a small number of categories but start to degrade as the category size increases. Additionally, the third method tries to make an adversarial image that will be classified based on a specified target category. The first two do not have this ability to specify a target category.

Nicolas et al [8] developed a practical method for attacking black-box machine learning systems. The work in the paper proves that, even if attackers do not have access to the model internals or its training data, AEs can be developed and used successfully. This work is a steppingstone for potentially attacking Tesla's Full Self-Driving (FSD) system. The paper achieves its goal by training a local model to substitute the target model. The local model is trained and tuned to mimic the results of the target model. After the local model is adequately trained it is used to develop AEs. These AEs are then fed into the target model. And indeed, the target model misclassifies the AEs. Although this proves black-box attacks are possible I believe attempting a black-box attack on Tesla's object detection will include a lot more complications and effort. Possibly making it not worth the effort. This would have been an interesting avenue to explore for my project, but I believe the work involved is beyond the scope of this project.

The last related work I will discuss is Cihang et al [2]. This paper introduces an AE for semantic segmentation and object detection. Object detection usually defaults to creating a bounding box around an object. Semantic segmentation attempts to create a mask around the object thus providing a more accurate prediction of the object's area in the image. The paper presents the Dense Adversary Generation (DAG) algorithm. Much like other AEs, DAG uses information of the model to misclassify objects as other objects detected by the model. Section 3.2 explains how DAG works in more detail. Additionally, the paper explores transferring adversarial perturbations from one model to another model. There is success in this exploration. This is why I chose the DAG AE. If I were to have any success in tricking Tesla an AE that is proven to have transferability would be a better candidate than an AE with unmeasured transferability.

3 Design & Implementation

In this section, I will discuss the first three milestones of my project.

3.1 Training Detectron2

Training a Detectron2 model was a fairly easy task to complete. While the work was easy it was very tedious. I will also take the time to point out that this is my first time dealing with and training a model. Previously, I had no experience in this area outside of the few research papers I read during this semester.

First, I started off with building my dataset of images. I used an open-source web crawler [9] to download the images I would use for training. The images were sourced from Flickr. After building my collection of images I had to build a common objects in context (COCO) annotation file. Detectron2 uses COCO annotation format for instance segmentation. The COCO annotation format has been used for at least six years. While it is widely adopted it can be too simplistic for some object detection algorithms. Luckily, no such modifications where needed in my project. I used an open-source program called Labelme [10] to trace the objects in the images. This was the first extremely tedious task. I spent roughly eight hours in total tracing roughly 200 bicycles and airplanes images. After you trace an image in Labelme it generates a json file that contains the polygonal annotation. This Labelme annotation needs to be converted into the COCO annotation format. Because Labelme and COCO are well-known there is an open-source converter [11] I was able to use. This made the process of converting the multiple Labelme annotation files into one COCO annotation file simplistic. With this step completed I was able to begin training.

Training a model with my computer was out of the question. It consistently crashed due to memory limitations. So, instead I used Google Colaboratory (Google Colab). Google Colab is a python playground. It allows a user to execute arbitrary python code through the Google Colab interface. It is especially well suited for machine learning because it allows instances of Google Colab to be ran on GPU infrastructure. Thus, I wrote the code to train my model within Google Colab and ran it from there. In Appendix A you can find the link to my Google Colab file.

This process was not without pitfalls. My first issue was where I hosted my model file. Hosting the file remotely enabled me to automatically download the file into the Google Colab environment. I originally hosted it on GitHub, but I quickly ran over my free data limits on GitHub. This left me with few options. Google Drive was one option but uploading and downloading the model file took too much time to be feasible. In the end, I resorted to manually uploading and downloading the model file into the Google Colab environment directly. Similarly, I had the same issue with my zip file of images. Again, I resorted to manually uploading those files. The unfortunate part of this decision is that the uploading process was far from perfect. Frequently the upload errored out and would have to be redone. Or the Google Colab session timed out. When the Google Colab session times out any files that were stored in the session are deleted. I quickly learned there was a fine craft to uploading a file successfully. This process demanded my attention during the upload. If I was not paying attention I would not know if my upload successfully completed or not. Plus, there was the threat of letting my session idle and time out. My second issue with Google Colab was, as briefly mentioned before, session timeouts. If the browser running Google Colab is not in focus on the foreground of the PC it will time out after a certain duration. This caused me to lose a lot of training time because mid-training I was susceptible to losing my progress with no way of creating a checkpoint. Again, I quickly learned the fine craft of running a session and avoiding timeouts. But that is not to say I was fool proof. Occasionally, I would accidently hit another window on my PC causing the browser to lose focus thus ultimately causing a timeout. Lastly, I discovered there are data limitations to Google Colab that I reached. Although, this roadblock was easily resolved by copying my Google Colab file to another Google account and working with this alternative account.

All in all, I spent roughly 48 hours training my model. Due to my inexperience and lack of knowledge of my selected adversarial example (AE) I had to retrain my model three times. My first mistake was I included an extra piece of polygonal information in my bicycle traces. As well as incomplete pieces of a bicycle e.g., a piece of a tire or handlebar. Appendix B Figure 1 has an example of these two mishaps. I only realized this mistake after I evaluated my model after training it for roughly 40,000 iterations. (40,000 was used as a baseline for my project because that is Detectron2's default.) After this realization, I went back and removed the troublesome data from my images and rebuilt my COCO annotation file and trained again. Due to my AE implementation I had to retrain two more times. In Section 3.2 I dive further into why this was needed. Ultimately, the second time could have been avoided but I made a careless mistake and retrained with an improper configuration set up which rendered the model completely useless. After fixing this careless mistake I retrained a 4th and final time.

With my model finally trained and working properly I could start building the AE and applying it to my dataset.

3.2 Building the adversarial example

As mentioned previously, I implemented the Dense Adversary Generation (DAG) algorithm presented by Cihang et al [2]. This AE considers all the targets and optimizes the overall loss function. The lower the value of the loss function the more confidently the model can predict on the targets. Thus, DAG will optimize the loss function for a given adversarial label. This will cause the model to predict the adversarial label instead of the correct target because the adversarial label will yield a lower loss function value. So, for example, in my project I had attacked images of bicycles (as well as airplanes). The loss function will yield a lower value for the adversarial label, airplane, compared to the correct target, bicycle. Resulting in the model predicting airplane when in fact it is an image of a bicycle. It is for this reason that I had to retrain my model with additional classes. I needed to include at least one other class for detection. I, ultimately, chose airplanes because they are simple to trace. Thus, reducing the work effort of building a new dataset of images. This decision was influenced by the fact that tracing the bicycle images took roughly 5 hours and at the point of this realization it was past Thanksgiving. I was quickly running out of time to complete my project.

I was fortunate enough to find an open source [12] implementation of DAG using Detectron2 on GitHub. Although, the code was not usable "out of the box" sort of speak. The code was design and written to attack Detectron2's model zoo. (A model zoo is a common way open-source framework organize their machine learning models.) After a few solid days of reading the code and working with the DAG implementation I was able to successfully get it to work with my model. The results were incredible and will be discussed in detail in Section 4.2. Suffice to say the DAG AE worked well.

3.3 Testing Tesla's Vision

Due to the nature of using the Model 3 this part of the project is a black-box test. I have no access into the inner computers or model that make up Tesla's object detection system. I am aware of some hacks that can get access into the OS behind the internal computers of a Tesla. But to what extent is unknown to me. Additionally, I would not dare risk voiding my warranty by hacking into my Model 3's computers. With that said my measurement for results is based solely on the visualization display on the screen inside the Model 3. Appendix B Figure 2 shows an example of this.

It is with great regret I have to say this part of the project was a failure. I attempted to visualize images of bicycles using three different approaches. The first approach was to display images full screen on a laptop and hold them next to the cameras on the car. The second approach was to print images on paper and hold them next to the cameras on the car. The third approach, as detailed in my proposal, was to print large-scale versions of the images and hang them. Then drive the Model 3 towards the image. None of these approaches work. I understood the first two approaches were likely to fail but I honestly thought the third approach had a chance at success. In section 4.3 I theorize why all these approaches failed.

To produce these large-scale images, I wrote a script that divides the picture into an equal number of rows and columns. In total an image was divided into 64 sub images. Once a picture was divided, I needed to print it out. I used Staples to print out these images. I printed these pictures on 11"x17" executive paper. I used the large size paper so I could construct an image large enough to match human scale. I used executive paper because it is a firmer paper that is less like to buckle and shift in the outside environment. At 0.78 cents per print a single full image cost $57. This cost factor limited me to only printing one image. Once the 64 images were printed, I cut any whitespace and taped them together to construct the large-scale image. With that said I printed an image that I knew the Tesla could recognize. It was a picture of a human on a bicycle that my Tesla recognized. I ran a small test to ensure my Tesla could recognize the subject. After Tesla recognized the subject, I use my phone to capture an image of the subject. I compared my image to the image the forward-facing camera recorded. Indeed, they were very similar. Appendix B Figure 3 display the three images discussed here.

4 Evaluation

In this section, I evaluate the results of my model and its average precision, the success of the Dense Adversary Generation (DAG) adversarial example (AE), a potential interesting finding involving the DAG AE, and I discuss why a possible reason why my Model 3 failed to recognize my images.

4.1 Model Evaluation

Before discussing the results, I will explain how object detection systems are measured. Object detection systems use a metric called average precision (AP) or means average precision (mAP). These two terms can be used interchangeably. Two compute the AP one must first compute the precision metric, recall metric, and the precision-recall curve. Precision is a measure of "when your model guesses, how often does it guess correctly?" [13] A common precision measurement is called intersection over union (IoU). Simply stated it is the overlap between two boundaries, the prediction boundary and ground truth boundary. Recall is a measure of "has your model guessed every time that it should have guessed?" [13] Consider an image with ten bicycles. If my model found only one bicycle but correctly labels it as bicycle, then my model has perfect precision but imperfect recall. It has perfect precision because every bicycle guess it made was correct. It has imperfect precision because only one out of ten bicycles were found. Using these two metrics a precision-recall curve can be generated. This is done by plotting the precision and recall as a function of the model's confidence threshold. AP is calculated as the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as a weight. [13] I omit the details of these formulas because I do not believe they bring much value to the paper. If such formulas interest you I direct you to reference [14]. Jonathan Hui goes into the details of the formulas used to compute these metrics.

Now I can begin discussing the results of my model. Appendix B Figure 4 and Figure 5 show the complete results of my model's evaluation. Figure 4 is the results of bounding box precision and Figure 5 is the results of instance segmentation precision. The different IoU values represent thresholds of varying levels of overlap between the boundaries. The words all, small, medium, and large refer to the size of these boundaries. As stated earlier, these boundaries are the prediction verses the ground truth. With that said this implies my model is evaluated on the fact that its predictions are compared against the ground truth (the training dataset). Appendix B Figure 6 and Figure 7. show the average precision per category for bounding box and instance segmentation, respectively. It is not surprising that the bounding box has a higher precision. This is because the bounding box is a less accurate prediction compared to instance segmentation. Appendix B Figure 8 hopefully clarifies this intuitive statement. To further elaborate on this point instance segmentation attempts to trace the complete subject of a given prediction. In the Appendix B Figure 8 we can see the edges of the trace has peaks and valleys. These peaks and valleys attribute to the lower AP values. The bicycle predictions performed worse than the airplane predictions. In many cases, and as showcased in Appendix B Figure 8, we can see the handlebars, seat, pedals, tires, and middle support bars have noticeable errors. The handlebars, seat, pedals, tires are incorrectly predicted. The middle support bars have chunks missing in the prediction. My best guess to explain this phenomenon is due to the fact that bicycles have gaps in them. These gaps cause the machine learning algorithm to account for data that is not part of the bicycle. I believe this attributes to a part of the inaccuracy when predicting bicycles. The airplanes do not have these gaps in their ground truth data which is another aspect that reinforces this statement. Although because my knowledge of machine learning is very sparse I cannot confidently back this statement.

4.2 DAG Evaluation

Due to an image size change during the process of applying the DAG algorithm I could not evaluate my attacked images using the AP metric. Because the image sizes changed the ground truth trace would not line up with the resized attacked image. This would certainly cause the data to be incorrect. I thought of resizing the images to their original size, but I was afraid this could degrade the efficiency of the AE. Instead, I decided to generate a concatenation of images and inspect them manually. I wrote a script that concatenated the original image, original predictions, DAG perturbations, attacked image, and attacked predictions. Appendix B Figure 9 is an example. During my manual inspection of the images, I found that 6 bicycle images and 22 airplane images failed to be misclassified by the AE. Appendix B Figure 10 is an example of a failed attack. To say in other words, the correct category was predicted on these 28 attacked images. At first, I thought maybe because of my consistent training data (e.g., very similar angles through the images) I had inadvertently discovered a defense against this AE. Initially I was excited because I thought I had discovered a novel defense. But upon further inspect of all of these images I realized the bicycles and airplanes were in varying poses. I concluded that due to the shallow nature of my dataset, only 191 images total, the AE did not have enough information from the model to properly perturb these images. That is to say that the perturbations that were applied to these 28 attacked images did not lower the loss function value of the attack category below the loss function value of the correct category. Furthermore, if this was indeed a practical defense, training a model on consistent data (bicycles and airplanes in a similar pose across all images) would cause the model to fail on images that had varying poses. Thus, the model would sacrifice its ability to accurately predict categories to defend against this AE. Which ultimately is not worth it. Essentially, what good is a model that can only predict a bicycle or airplane from a sideview and not a front, rear, or any other view?

4.3 Tesla Evaluation

I have concluded that Tesla's object detection algorithm uses more than just the car's cameras. Appendix B Figure 11 shows an image of the Model 3's sensors. There are two types of sensors that are not cameras: 12 ultrasonic sensors and 1 radar. I knew the Model 3 had the ultrasonic sensors, but I only thought their use was to detect objects when in park, reverse, or moving in tight spaces. During any of these three scenarios the screen changes how it visualizes information around the car. Instead of models of objects around the car, Appendix B Figure 2, there are color lines and a range indication. These lines range from gray to yellow to red and indicate the proximity of an object from the car. Appendix B Figure 12 is an example of this visualization. Searching for additional information on these sensors does not yield much information. References [15] and [16] loosely discuss these sensors. Barely giving any insights on how they are used. One interesting piece of information is that the radar wavelength passes through fog, dust, rain, snow, and undercars and plays an "essential role" in detecting forward objects. So, the question is, if the forward camera is unreliable in these conditions how does the radar correctly identify forward objects? I know from personal experience in these conditions the visualization display correctly displays the model of what is in front of me. E.g., a car model instead of a SUV or truck model. Also, it is not mentioned, what about sunlight shining directly into the forwarding cameras? Certainly, that has to have a negative effect on car's ability to "see" objects in front of it. I have concluded that the information gathered from these sensors is built into data similar to point-cloud data. Real quickly, image a thousand tiny balls creating the shape of a human, this is the idea of point-cloud data. I believe this point-cloud data is used to reinforce the object detection algorithm. Or possibly serve as a backup if the forward-facing camera is unreliable due to external conditions. Now when I review my testing process, I understand why it failed. If my theory is correct the point-cloud data to reinforce a bicycle in front of my car is not there. The radar is hitting a solid object which is the large-scale image. Without a doubt if the radar is somehow calculating depth then the depth of a person on a bicycle and a hanging solid object will be different. This similar idea can be applied to the ultrasonic sensors as well. But this all relies on the fact that my theory is correct. So, at best this statement is conjecture. It is totally plausible that the object detection algorithm is strong enough to differentiate the different between an image and a real-life object.

5 Conclusion

I believe the first part of my project is very strong. Not only that but I learned a lot from going through the whole process of machine learning; data gathering to model evaluation. The only thing that is missing from this experience is the building of such a system, which is beyond the scope of this project and class. Regardless, a lot of lessons were learned on this journey and even so there is a ton more to learn in the world of machine learning. My greatest appreciation is this project forced my hand into using machine learning and stepping into an unknown area of knowledge. Next semester I am taking Applied Machine Learning. I am excited to have prior experience before entering this class. Hopefully, the lessons learned from this project will apply in that class.

While the first half of my project was strong (in my opinion) I am not blind to the fact that my second half fell extremely short in the expectations of this class. I failed to adequately complete what I set out to accomplish as well as discover any novelties. In retrospect I was too focused on incorporating my Model 3 into my project. I remember Professor Wang mentioned instances of researchers attacking object detections and that is when I became infatuated with this idea. Ultimately when I realized my Model 3 was unable to visualize stop signs, I should have abandoned the idea of testing Tesla's vision completely. Additionally, I could have done some preemptive research and tests to see how feasible my methods of testing Tesla's vision are. This could have shown me that my project was designed to fail. With that knowledge I could have thought of another project that could have led to novel discoveries or work. Regardless of how the project ended I have a firm belief it was still extremely beneficial to me.

References

[1] Yuxin Wu, Alexander Kirillov, Francisco Massa and Wan-Yen Lo, & Ross Girshick. (2019). Detectron2. https://github.com/facebookresearch/detectron2.

[2] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Yuyin Zhou, Lingxi Xie and Alan Yuille, "Adversarial Examples for Semantic Segmentation and Object Detection," 2017 IEEE International Conference on Computer Vision (ICCV), Venice, 2017, pp. 1378-1387, doi: 10.1109/ICCV.2017.153.

[3] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Florian Tramer, Atul Prakash, Tadayoshi Kohno, and Dawn Song. 2018. Physical Adversarial Examples for Object Detectors. In 12th {USENIX} Workshop on Offensive Technologies ({WOOT} 18).

[4] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. 2017. Robust Physical-World Attacks on Machine Learning Models. CoRR abs/1707.08945 (2017). arXiv:1707.08945 http://arxiv.org/abs/1707.08945

[5] Yue Zhao, Hong Zhu, Ruigang Liang, Qintao Shen, Shengzhi Zhang, Kai Chen. 2019. Seeing isn't Believing: Towards More Robust Adversarial Attack Against Real World Object Detectors. In 2019 ACM SIGSAC Conference on Computer and Communications Security ({CCS} 19)

[6] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In ICLR, 2015.

[7] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In ICLR Workshop, 2017.

[8] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, & Ananthram Swami. Practical Black-Box Attacks against Machine Learning. In 2017 ACM on Asia Conference on Computer and Communications Security.

[9] amineHorseman (2018, Augth) *images-web-crawler* GitHub. https://github.com/amineHorseman/images-web-crawler/

[10] wkentaro (2020, September 29th) *labelme* GitHub. https://github.com/wkentaro/labelme

[11] Tony607 (2019, July 30th) *labeme2coco* GitHub. https://github.com/Tony607/labelme2coco/blob/master/labelme2coco.py

[12] lemonwaffle (2020, August 2nd) *detectron2-1* GitHub. https://github.com/lemonwaffle/detectron2-1

[13] Jacob Solawets (2002, May 6th) *What is Mena Average Precision (mAP) in Object Detection?* Roboflow. https://blog.roboflow.com/mean-average-precision/

[14] Jonathan Hui (2018, March 6th) mAP *(mean Average Precision for Object Detection)* Medium. https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173

[15] *Autopilot* Tesla. https://www.tesla.com/autopilot

[16] *Autopilot and Full Self-Driving Capability* Tesla. https://www.tesla.com/support/autopilot

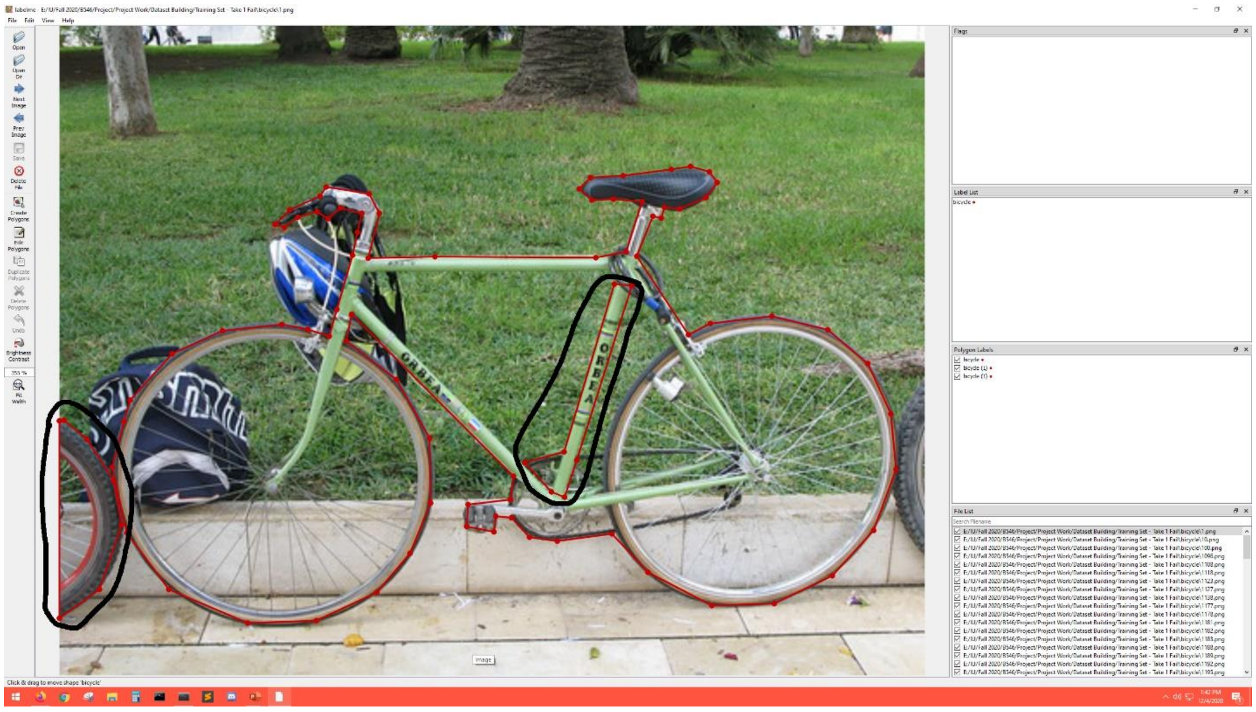
Appendix A

Google Colab Link: https://colab.research.google.com/drive/1FalDA23baaEK-ue8qCXgIbDaWtMnmOvg?usp=sharing

GitHub Link: https://github.com/EthanBehar68/TestingTeslasVision

Appendix B

Figure 1: An example of extra polygonal information, circled in black, included in the first pass of my dataset.



labelme - E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\1.png

Label List: bicycle

Polygon Labels:
- bicycle
- bicycle (L)
- bicycle (1)

File List:
```
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\1.png
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\10.png
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\100.png
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\1096.png
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\1108.png
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\1118.png
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\1123.png
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\1127.png
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\1138.png
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\1177.png
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\1178.png
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\1181.png
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\1182.png
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\1183.png
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\1188.png
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\1189.png
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\1192.png
E:/IU/Fall 2020/B546/Project/Project Work/Dataset Building/Training Set - Take 1 Fail\bicycle\1193.png
```

Click & drag to move shape 'bicycle'

1:42 PM
12/4/2020

Figure 2: An example of Tesla's object visualization interface. Fun fact, my bicycle, as seen in this picture, was stolen two days after I took this photo.

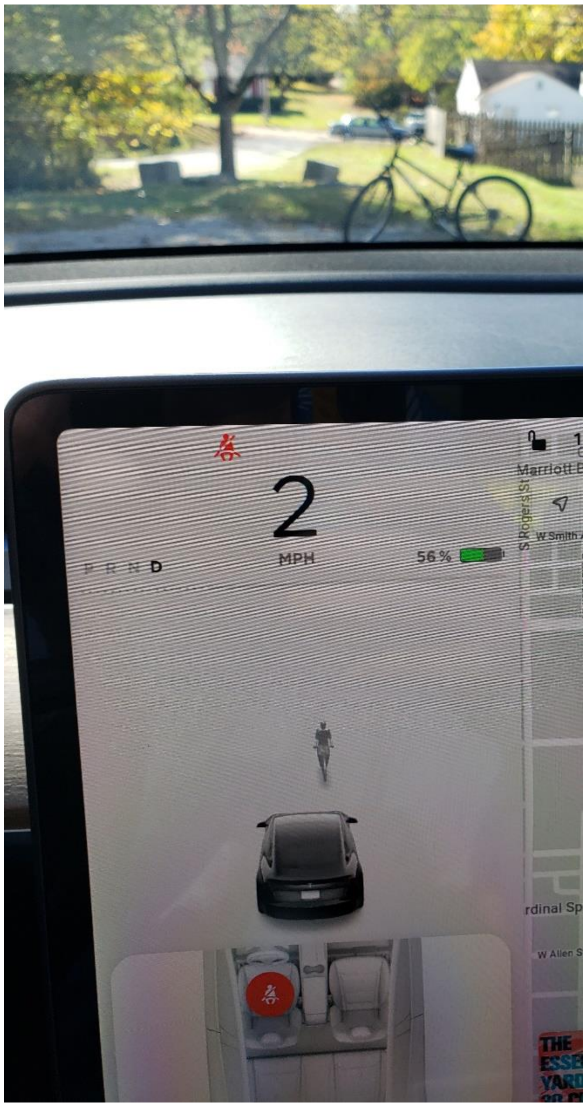


Figure 3: Left: An image taken from my Model 3's forward camera. This image is cropped from the original video feed. Center: An image taken from my phone resized. Right: Large-scale image constructed using 64 printed images. I did my best to ensure the distances of the photos were the same. I measured where the forward-facing camera was when it detected the subject of the picture. From that measure I took the photo with my camera. You can see the tape measure in the right photo on the left side of the concrete.

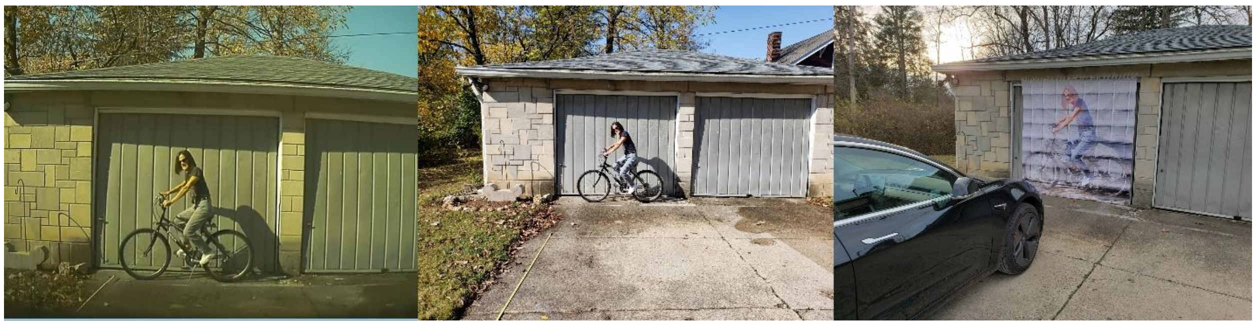


Figure 4: Bounding box average precision of my model.

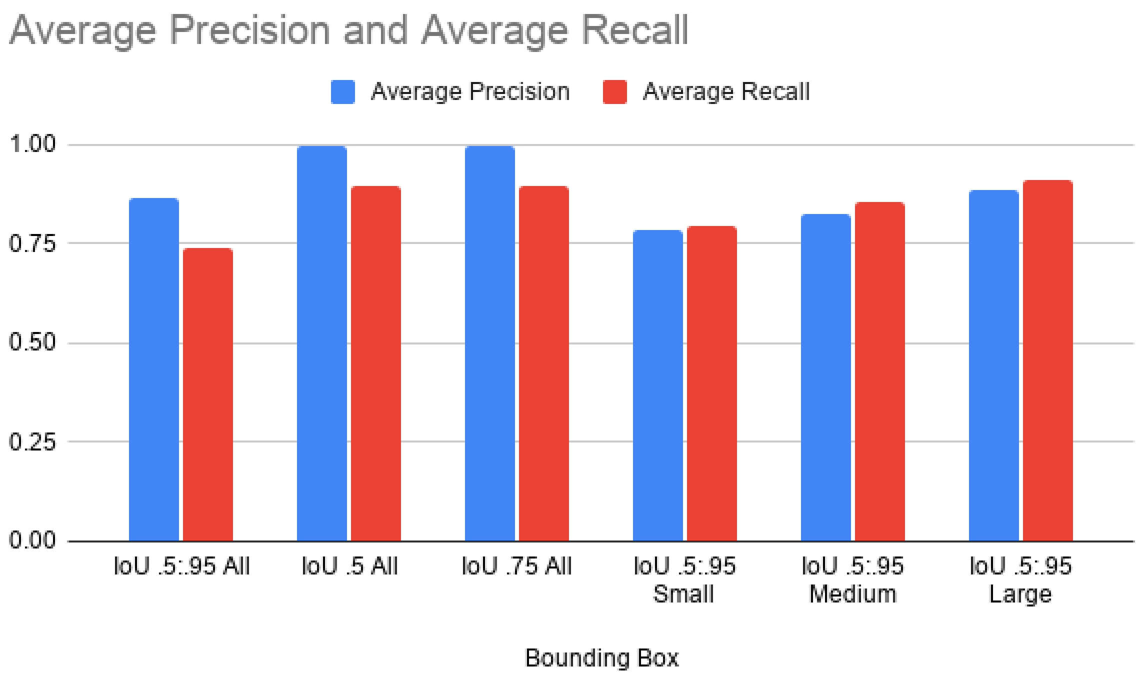


Figure 5: Instance segmentation average precision of my model.

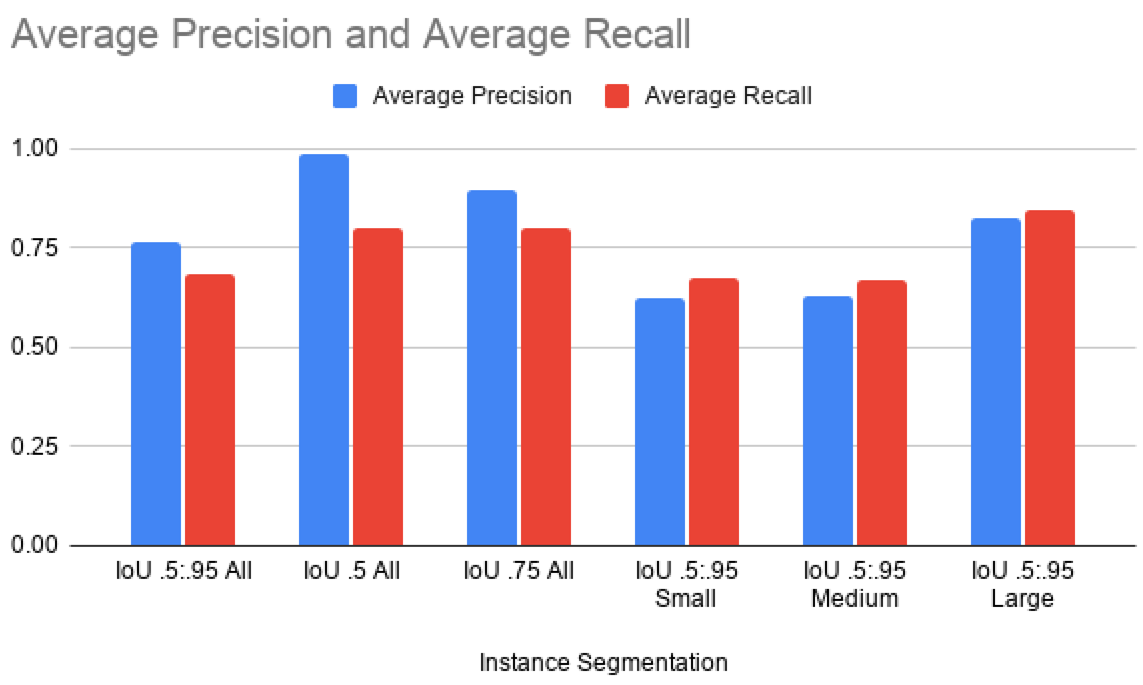


Figure 6: Bounding box average precision results per category of my model.

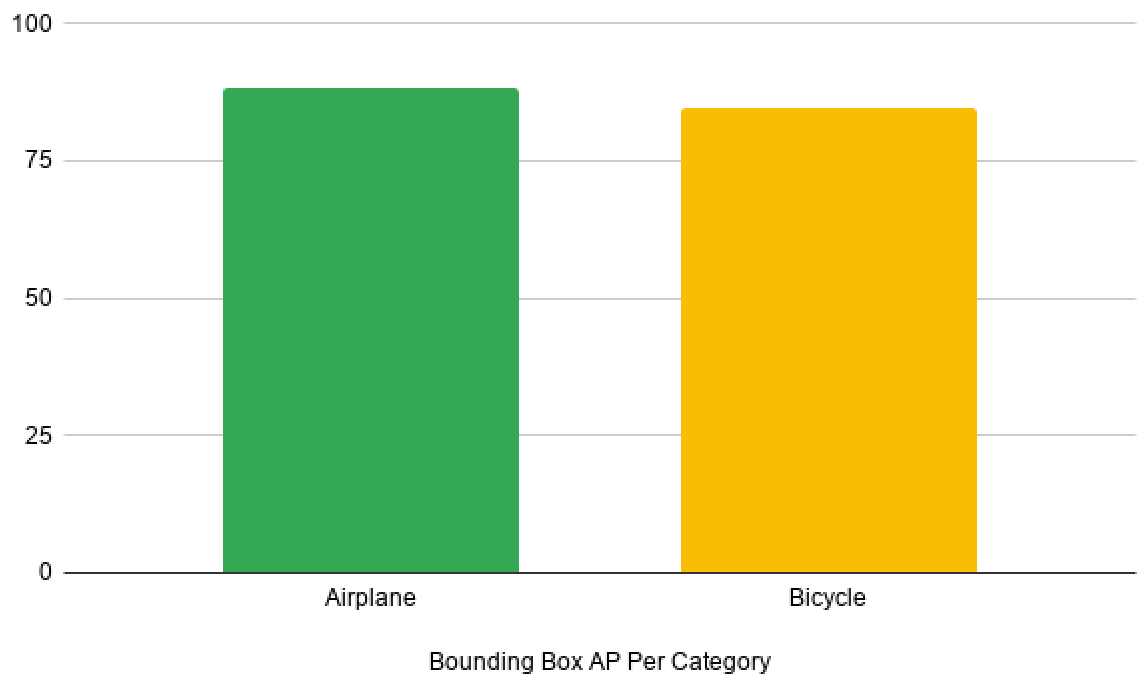


Figure 7: Instance segmentation average precision results per category of my model.

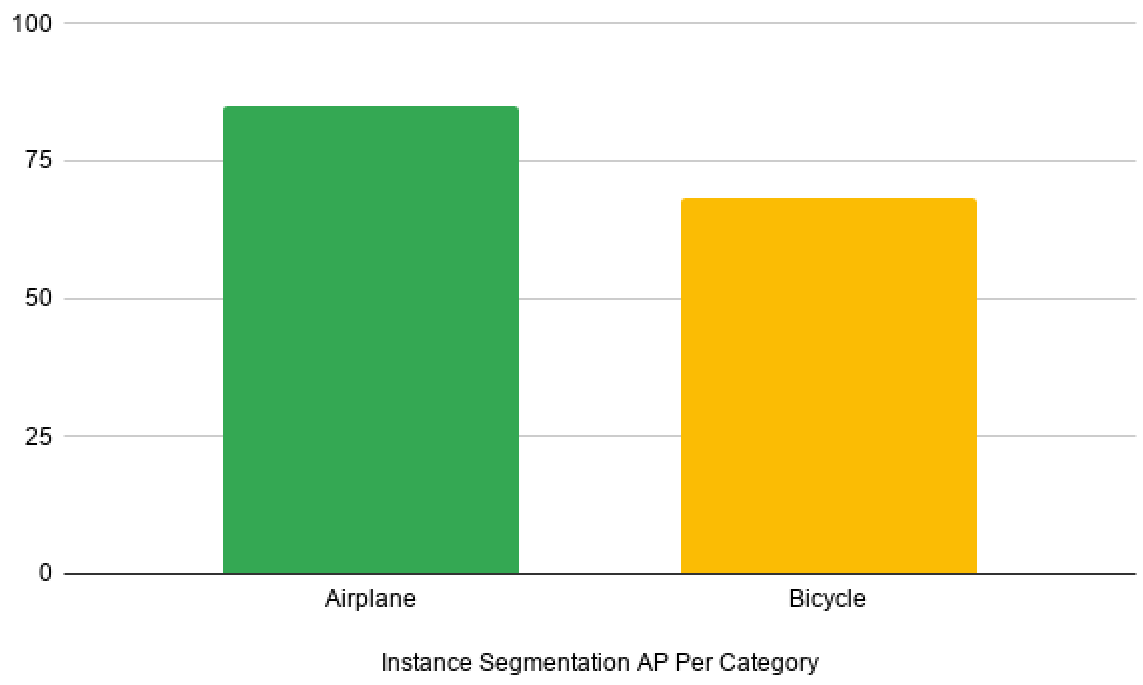


Figure 8: An example of bounding box and instance segmentation predictions from my model. Here we can see why instance segmentation has a lower score. Compared to the trace in Figure 1, minus the black circled areas, the instance segmentation has noticeable errors.

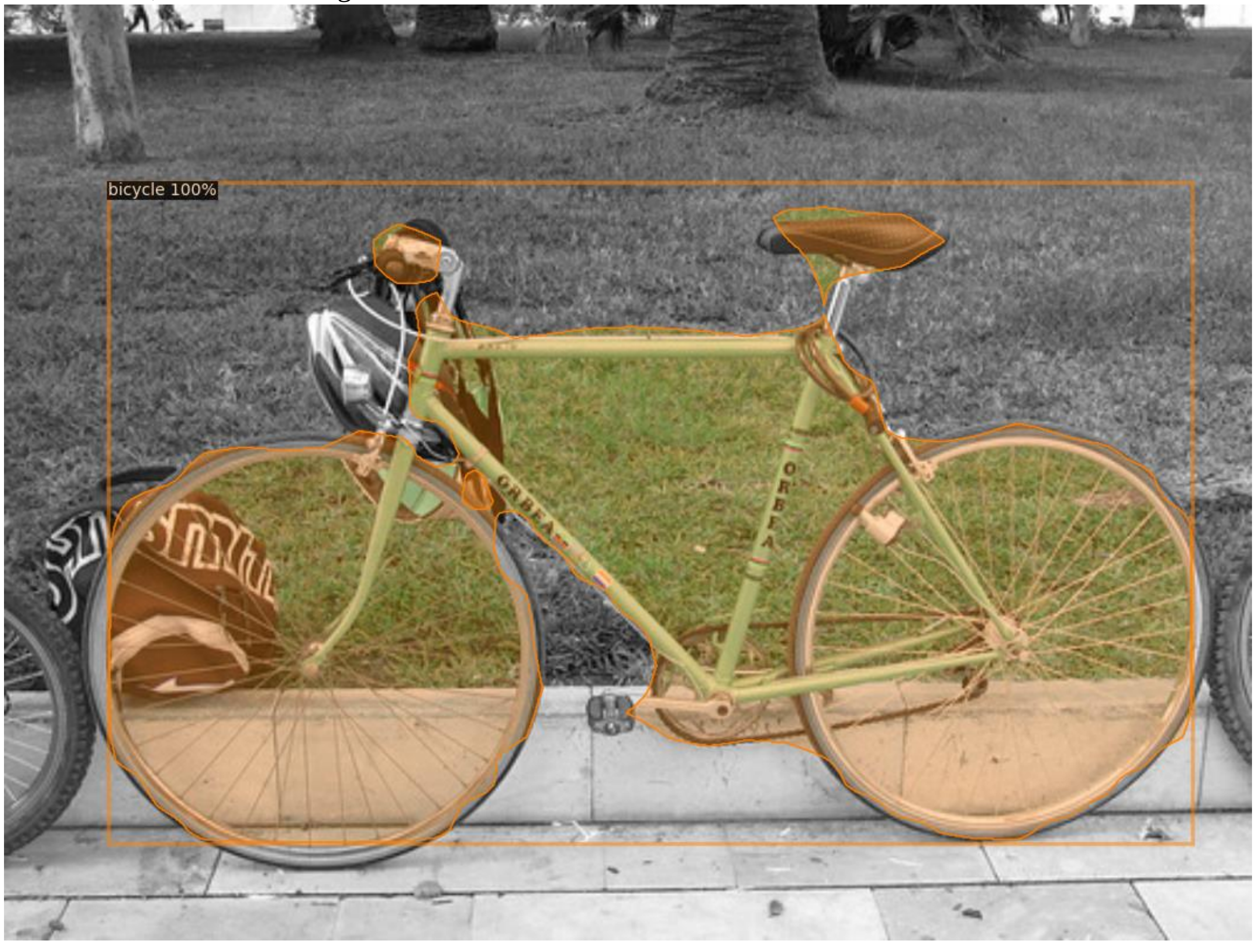


Figure 9: From left to right: original image, original prediction with my model, DAG perturbation applied to original image, attacked image, attack image prediction with my model. Here the AE successfully causing my model to misclassify the bicycle as an airplane.

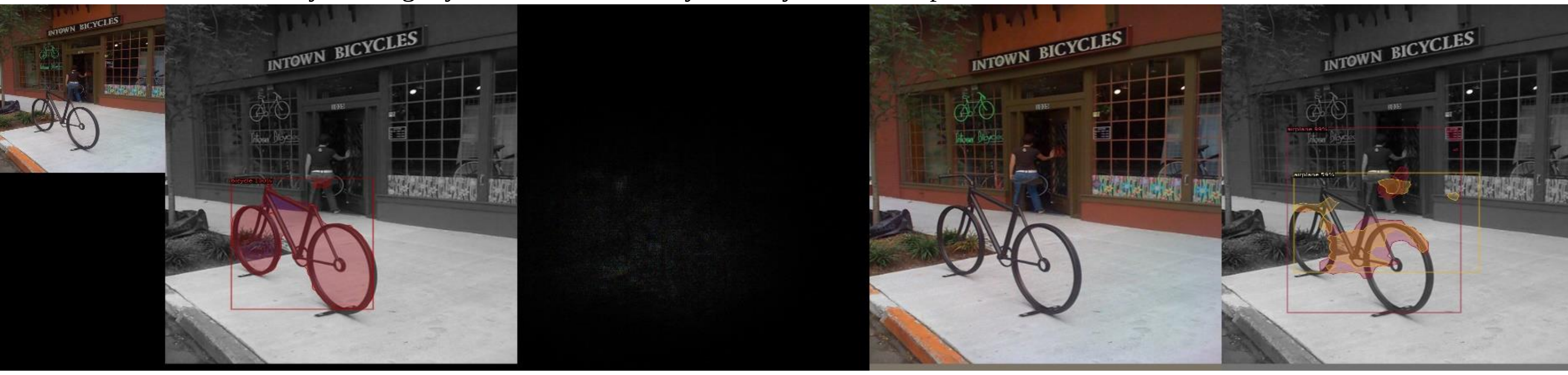


Figure 10: From left to right: original image, original prediction with my model, DAG perturbation applied to original image, attacked image, attack image prediction with my model. Here the AE failed to cause my model to misclassify the bicycles. Oddly enough, the AE improved the prediction of the bicycles.

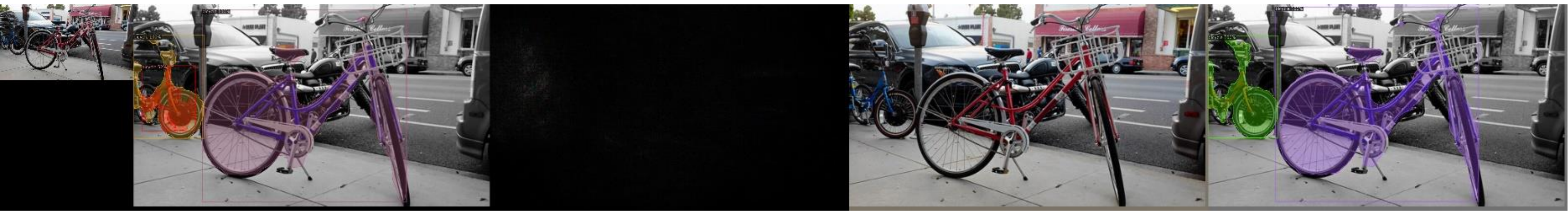


Figure 11: Tesla Model 3's sensors. Image taken from [15].

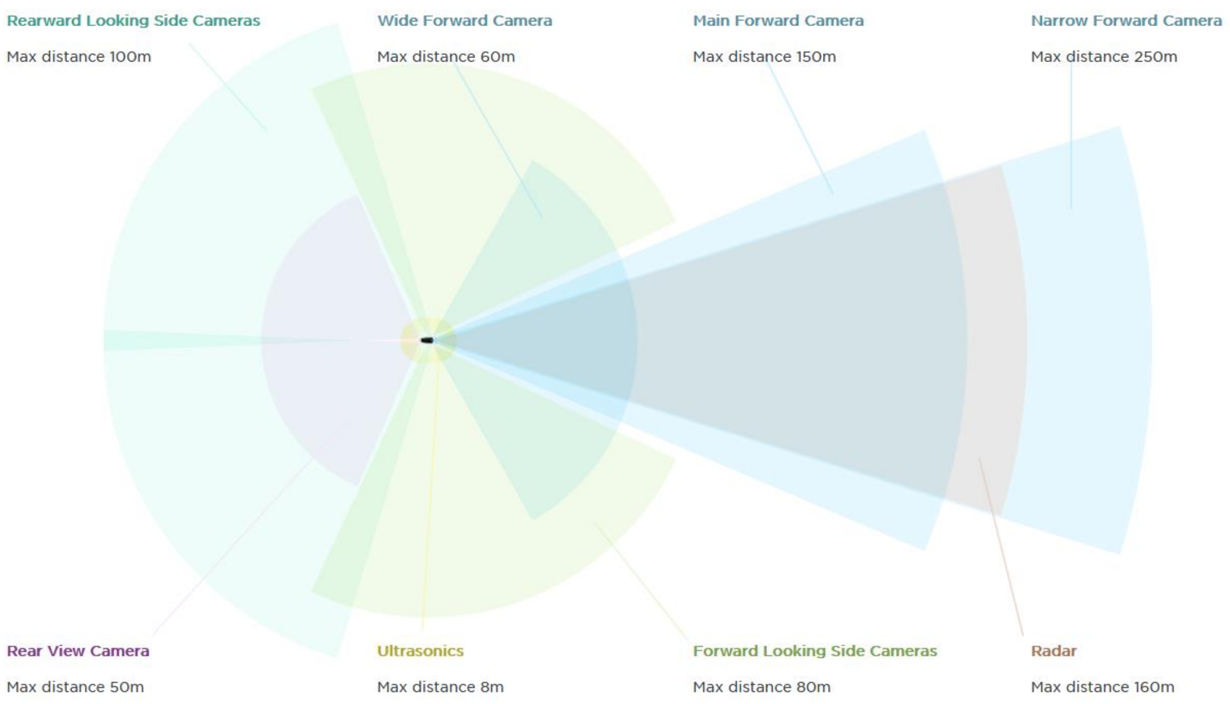
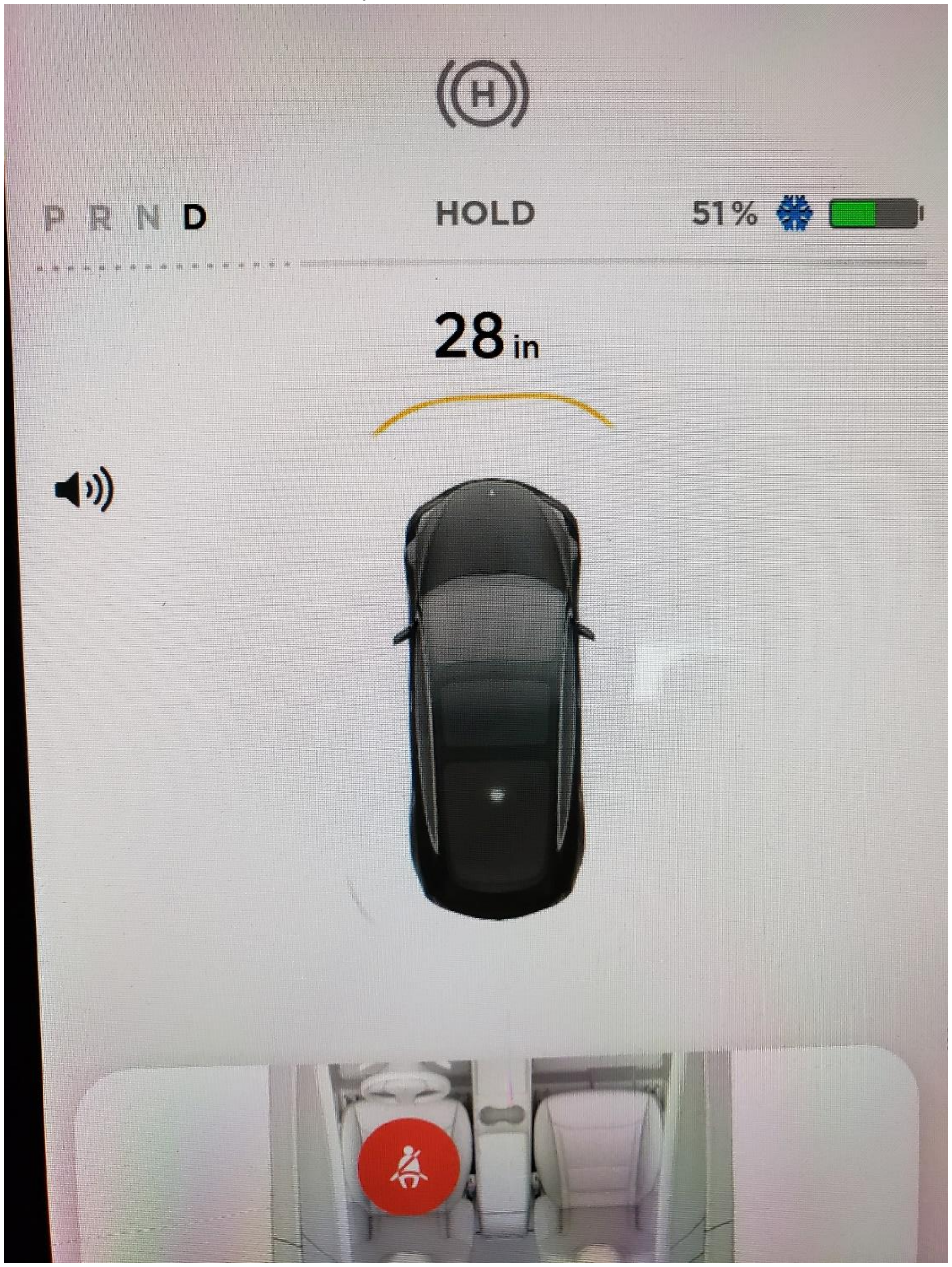


Figure 12: Tesla object detection when in park, reverse, or moving in tight spaces. It is my belief that the lines are the car are measured by the ultrasonic sensors and not the cameras.


P R N D
HOLD
51%
28 in