HIGH SCHOOL EDITION
Journal of Student Research

Volume 10 Issue 2 (2021)
AP Research August

Impact of Model Architecture Against Adversarial Example's Effectivity

Vihan Karnala[1] and Marianne Campbell[1#]

[1]Milton High School, Alpharetta, GA, USA
[#]Advisor

ABSTRACT

The purpose of this study is to gain an understanding of the impact of model architecture on the efficacy of adversarial examples against machine learning systems implemented in self-driving applications. Prior research shows how to create and train against adversarial examples in many use cases; however, there is no definite understanding of how a machine learning model's architecture affects the efficacy of adversarial examples. Data was collected through an experimental setting involving end-to-end self-driving models trained through behavioral cloning. Three model types were tested based on popular frameworks for machine learning algorithms dealing with images. Results showed a statistically significant difference in the impact of adversarial examples between these models. This means that certain model types and architectures are more susceptible to attacks. Therefore, the conclusion can be made that model architecture does impact the efficacy of adversarial examples; however, this is potentially limited to closed-loop, end-to-end systems in which algorithms make the entire decision. Future research should investigate what specific structure within models causes increased susceptibility to adversarial attacks.

Introduction

Machine learning is the process in which a computer develops its own method for completing a task by recognizing patterns [1]. Recent advances in machine learning research have allowed companies in the private sector to create technology such as facial recognition-equipped cameras and predictive tools for the stock market. This technology has wormed its way into people's everyday lives with its various applications such as those in social media and voice-based assistants. The integration of machine learning algorithms into the physical world is increasing, especially with the billions of dollars being poured into machine learning research.

Specifically, artificial neural networks (ANNs), a type of machine learning algorithm that has many layers of artificial neurons, have gained immense traction in various machine learning applications in recent years [2], [3]. However, this type of algorithm has an issue. Adversarial examples, defined as carefully curated inputs that fool machine learning systems, are highly effective against ANNs [4], [5]. As these types of machine learning algorithms gain popularity and influence over human lives, we must look at how they can be protected in the event of malicious attacks. Adversarial examples pose a real problem to machine learning systems as they are often created to be imperceptible to humans. Although there is plentiful research on machine learning and adversarial examples individually, the factors that impact how well adversarial examples can impact a machine learning system have seldom been researched. I will be using virtual representations of self-driving vehicles with various model architectures to record the impact adversarial examples have on the vehicle's driving efficacy. My goal will be to answer the question: to what extent does model architecture impact adversarial example effectivity in self-driving vehicles?

History of Machine Learning in Self-Driving

Early computer vision applications for autonomous vehicles came with the introduction of computational approaches to edge detection, as seen in the paper by John Canny [6]. His paper from 1984 discusses methods to find edges from raw image data through fixed transformations of the pixels. Initial methods created for picture manipulation and feature extraction led to a race to produce autonomous vehicles. Kanade, Thorpe, and Whittaker, who researched early implementations of computational approaches to self-driving vehicles, discuss their attempt to create a self-driving vehicle that utilizes computational methods to find optimal paths along sidewalks [6]. Although their implementation of driving algorithms does not have any machine learning aspect, their research showed that real-time, completely autonomous navigation was possible and feasible.

Later research and the DARPA Grand challenges brought attention to computational self-driving, with universities competing against each other to build completely autonomous vehicles that could traverse deserts and urban cityscapes. These challenges showed the world that self-driving vehicles were possible and relied heavily on machine learning algorithms to make decisions.

Current Implementations

The field of machine learning and its applications has grown vastly over the past 30 years, with modern-day cars featuring assistive self-driving capabilities that allow the vehicle to drive alongside traditional human-driven vehicles. In their self-published paper, researchers from Nvidia discuss an approach to self-driving vehicles through the use of a monocular system (single camera) at test time and a convolutional neural network. A convolutional neural network, or CNN, is a type of ANN that can be applied to computer vision problems. CNNs have layers that scan raw pixel data from an image to recognize features. This feature recognition is normally followed by layers of fully connected neurons that take the recognized features and outputs a classification that correlates to them [3]. The research from Nvidia shows that using a single camera for driving a car on roads can operate with human intervention only needed approximately 2% of the time during a 22-minute drive [3].

Over recent years, research with CNNs and their applications in image classification has allowed researchers and companies to realize the true potential of this technology. One of the most popular examples of CNNs becoming mainstream is their application in the 2010 ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where they performed better than any preceding model at accurately classifying images. This challenge trained algorithms on millions of images that fit into 1000 categories. Some of these categories are dog, train, strawberry, and so on. This application of machine learning is similar to the classification that occurs during the driving of a vehicle. While driving, images from cameras around the vehicle are classified into the steering angle outputs, whereas in the challenge, they would be in named categories.

Recently, there has been advancement in using end-to-end models to drive vehicles [7]. These models are different from current commercial self-driving products as these solutions use a single CNN to take inputs from a forward-facing camera and directly produce a corresponding steering angle. Whereas current self-driving solutions have a mixture of neural networks and mathematical transformation to ensure the models have an accurate understanding of the vehicle's surroundings [8]. Seeing that there is a shift towards end-to-end models in the industry, we must discuss the potential for malicious actors against this technology. While machine learning technology is still in its relative infancy, there are few attacks against the technology outside of academic settings. However, as the technology gains popularity and is increasingly integrated into our society, there will be more instances of attacks against it.

Adversarial Examples

Malicious inputs to any machine learning system are commonly referred to as adversarial examples. These examples are carefully perturbed input samples aimed to mislead detection and will be the primary focus of my research [9]. An important example of adversarial examples as they will apply to fool self-driving classifiers can be seen in the paper by Szegedy et al., who describe adversarial examples in the context of image-based CNNs. Their paper discusses the creation of adversarial examples and how they impact models trained on classifying images [5]. Creating adversarial examples for machine learning models has become easy, with many methods being able to produce sophisticated attacks that are able to significantly change the output of a model while also being imperceptible to the human eye. However, there is no clear understanding as to what causes a machine learning model to be susceptible to adversarial examples in the first place. I plan to answer this by finding if there are architectures that are susceptible to adversarial input. This will show that certain models are impacted by adversarial examples.

Experimental Framework

Simulator Platform

In order to create an environment where the impact of adversarial examples can be quantitatively measured against model architecture, there must be a way to run repeatable experiments to collect data on the behavior of a self-driving vehicle. Running an experiment in the real world with a self-driving technology retrofitted to a car is not feasible. Instead, an open-source self-driving vehicle platform, Donkey Car, was chosen. Donkey Car allows the creation and implementation of various machine learning systems to a physical car platform [10]. Additionally, Donkey Car includes a high-fidelity virtual simulator called Donkey Simulator. This virtual simulator option was chosen as the virtual environment allows for repeated testing under the same conditions every time and is more accessible compared to other driving simulator offerings such as TORCS (The Open Racing Car Simulator) and CARLA (An Open Urban Driving Simulator) due to its minimal startup cost and associated learning curve. The usage of a simulator to test self-driving vehicles in an adversarial setting is well documented and proven to be effective [11], [12].

Furthermore, the Donkey Simulator returns telemetry values such as position and cross-track error, which allow for quantifying the efficacy of self-driving models. Donkey Car is primarily written in Python and communicates to the virtual simulator to receive telemetry and send actions (steering angles and throttle values). All data that is produced from the movement of the virtual vehicle is saved to the local computer, including images from the vehicle's point of view and the generated adversarial examples. Additionally, the Donkey Car platform is flexible to adaptations for research in adversarial examples. As the experimenter has already had experience with the platform and Python prior to the current research, Donkey Car was the optimal platform for this research.

Driving Model Architecture

Model architecture refers to the structure of the machine learning algorithms that will be used to drive the virtual vehicles. The architecture consists of various layers with varying depths that perform calculations on the input field (images of virtual road) to produce an output (steering angle). To test the impact of changing architecture versus adversarial examples, three model architectures were chosen. The three model architectures are Dave-2, VGG, ResNet [2], [3], [13]. The differences between these models lie in the sequence of layers that make up the models. Each model is slightly different in how the inputs are mathematically transformed into the outputs. The Dave-2 and VGG architectures are the most similar, with VGG being slightly shallower (less layers between input and output). The main difference between these models is the number of filters each layer has with VGG having more layers than the Dave-2 model. Increased filters means that each layer in the model outputs a greater amount of information to the next layer.

The ResNet model on the other hand introduces a mechanic that is not present in either of the other models in which a residual of previous layers outputs is passed to the next residual block. This means that each block of layers has direct connections to not only the previous block but also the block before it. Essentially, there is more interconnectedness between the blocks and therefore the layers.

The three original model architectures were first compressed to decrease the number of trainable parameters. Parameters are variables that allow layers to hold meaning. The original models have trainable parameters ranging from 250,000 to 2,000,000 parameters. The three models were compressed down to approximately between 250,000 to 500,000 trainable parameters to ensure that the training and testing phase are able to run efficiently on the researcher's computer. The varying values of parameters have little impact on the models as the number of parameters has a minimal impact on outputs [14]. A detailed model architecture is available in appendix A. The Dave-2 model architecture was chosen for its simple implementation and its proven efficacy in self-driving applications [3]. The VGG and ResNet model architectures were chosen for their frequent use in machine learning applications involving image recognition [2], [13]. Although these models were originally created ILSVRC, their application of model architecture for self-driving is assumed to be effective as end-to-end self-driving is parallel to image recognition, the only difference being instead of outputting a categorical value (type of object seen in the image), the self-driving application requires a continuous value (steering angle).

Since adversarial examples have the power to change the values of the output, only the steering angle is tested in this experiment. This is to ensure the virtual vehicle is always moving and does not stop during the testing phase, as adversarial inputs could possibly output negative or zero throttle values. A constant throttle value is given so that the vehicle consistently traverses the same distance between timestamps.

Training

The next step in testing adversarial examples is the creation of the target models that will be attacked. Since the model architecture is finalized, we must look at training the model to drive. Behavior cloning, a method of teaching a machine learning algorithm how to complete a task, was implemented for the training of this model. This method was chosen because of its ease of implementation and its ability to "yield optimal results efficiently" [15]. Behavior cloning allows for pre-collected data from a human expert to be treated as a ground truth dataset. Essentially, throughout its learning process, the models will find features from the images presented to it to justify an expert's outputs. Behavior cloning requires human-driven data collection for the expert dataset. The data used for training the target model includes an image from the vantage point of the car and the steering angle given by the human expert. Once the data of approximately 30,000 image and telemetry pairs were collected by driving a virtual car around the "generated road" track in the Donkey Simulator, the training phase of the driving model is started.

During training, the model learns from the data fed to it by utilizing a loss function. The loss function for the driving models is mean squared error, a statistic that quantifies the difference between the expected output and the output produced by the model. The expected output while training the driving model is the human expert's steering angle. While training, the loss function works in tandem with the optimizer function that looks at the loss function's output at a timestamp and decides the direction the weights must move to minimize the loss function at that specific timestamp. Repeated movement of the weights to minimize the loss function allows for the training of the model.

A physical representation of this movement is a gradient descent problem in a high-dimensional space. The loss function defines the surface that spans the space with mountains and valleys, and the optimizer decides what weights should be adjusted in order to find a global (or local) minimum across the surface. The optimizer chosen for this application is Adam for its prevalence in applications to convolutional neural networks and end-to-end systems [16]. While training, the model slowly decreases its loss and creates a viable generalization for the data given to it. This produces a model that is capable of driving solely on images fed to it through a virtual camera. The training method was stopped when the loss did not make a change of greater than 0.0005 units for more than five epochs (iterations through training data).

Training the machine learning models this way is similar to how a student learns in a class. The loss function can represent a test that returns a grade. The loss function then informs the teacher or tutor, who can be compared to an optimizer, on what the student needs more information on and can personalize lesson plans based on the questions that the student got wrong.

Adversarial Example Generator

Finally, in order to create varying adversarial example generators, an implementation of the Fast Gradient Sign Method (FGSM) was chosen [17]. This framework allows for the generation of adversarial patterns through a white box attack. This type of attack means that the model and all of its weights are known to the adversarial example generator. FGSM, by definition, shifts the pixels in the image towards the direction of greatest increase of the loss function surface discussed earlier (gradient ascent to counter the gradient descent in training). This shift of pixels in the generated adversarial example is hopefully enough to cross a decision boundary for the driving model. A decision boundary in this application would be the difference in pixels between two distinct angle outputs. This method constitutes an untargeted attack. An untargeted attack is suitable for this application because any output other than the expected output in a self-driving car has the potential to be catastrophic. FGSM constructs adversarial images by watching an image go through the model and recording the pixels that have the most influence over the output. By maximizing those pixel values, an adversarial pattern is generated. This pattern is then added to the original image to create the adversarial example.

FGSM needs complete access to the model that is being attacked. This attack type is largely impossible to achieve in the real world against existing self-driving solutions as base-level access is needed to the model, the architecture, and the vehicle that is being attacked [17]. Therefore, this attack is not possible to recreate in the real world; however, it can still show the impact of what changing model architecture does on the effectiveness of the adversarial images.

Testing

The simulator will handle the testing of the models. Each model type will have runs with and without an adversarial attack. When run with an attack, an adversarial image will be injected every five frames. The testing will be allowing the driving model to drive around the virtual track for 20 minutes (approximately 10,000 total frames and 2000 attacks). The data will contain adversarial images and the behavior (how it moves) of the vehicle as it drives along the track. The impact of model architecture on the efficacy of adversarial images will be determined at the image level and the behavioral level. At the image level, the changes between images will be measured and quantified by a Euclidean distance (L_2-Norm). The changes in behavior will be measured through cross-track error (CTE). This value is irrespective of the driving model and is a measure of the distance from the center of the right lane (the simulator is on a two-lane road) to the vehicle. The model architecture that is the most protected against adversarial examples will have the lowest overall variance in its CTE.

Analysis

In order to analyze the collected data, there will be two rounds of F-Tests on the CTE values from each run. This statistical test allows for the measurement of the change in standard deviation between two datasets. The first round of F-Tests will be between runs of the same model with and without an attack. This round will prove that in the case of each model type, the attack has a statistically significant difference in the CTE values. This can be assumed as between the runs with and without an attack; the only change is the presence of an adversarial attack. The second round of F-Tests will compare the collected CTE values between the attacked runs of each model. This will show that

there is a difference in the impact of adversarial examples caused by each model architecture as model architecture is the only thing that is changed between these runs.

Results

The hypothesis states that model architecture has a significant impact on adversarial examples' efficacy. The data collected includes six runs of three models, both with and without an attack. Within the data, every frame passed from the simulator to the driving model was collected and saved. Additionally, values passed back from the simulator describing the car's location in relation to the track were also collected. Finally, if an adversarial attack was generated for a frame, both the adversarial image and the original image were saved. The first of these data points is the position of the vehicle within the virtual environment. The position was collected to assist in communicating the differences in the vehicle's path between runs and models.

Figure 1 clearly shows the increased variance in the way the vehicle moves for the ResNet model. The attacked run (left graph) shows wider turns and an overall increase of visual deviation from the non-attacked runs. ResNet depicted the most difference in position between runs. While the position of the vehicle during runs helps us visualize the difference between a normal and attacked run, it is not helpful in quantifying how each model architecture is impacted by adversarial examples.

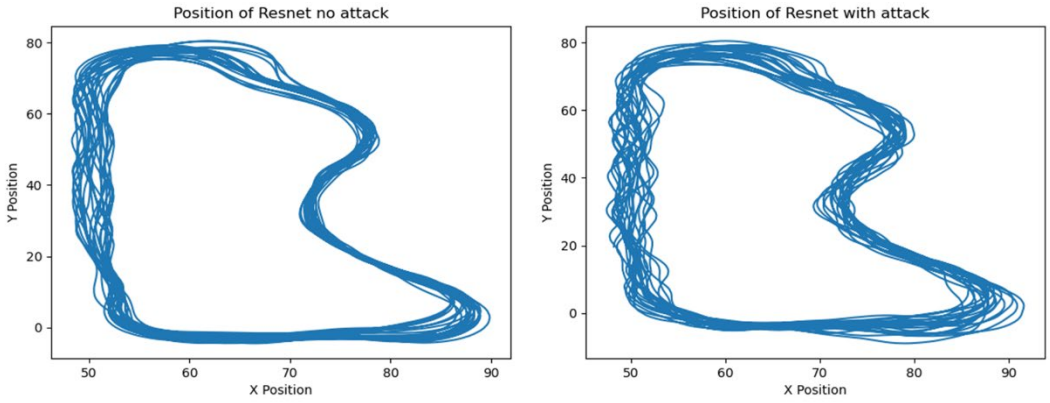

Fig. 1. Depicts the position values of the ResNet model runs with and without attacks.

Fig. 2. Above is the original image. Below is an example of an adversarial example created for the Dave-2 architecture. The difference between the two images is the addition of an adversarial pattern in the lower image.

Image Norm

Another data point calculated from the data recorded in the runs is the average image norm. This value is a measure of how much each adversarial image changes in relation to its original image. Fig 2 shows a sample of adversarial and original images. Image norm is calculated by subtracting the adversarial image's pixel values for red, green, and blue from the original images, squaring the difference, summing the squared difference across the entire image, and then taking the square root. This value measures the difference in each adversarial image holistically. Since all of the settings for the creation of the adversarial examples are constants between each run, we can assume that the reason for the change in average image norm is due to the varying model architecture. A larger image norm also means that there is a comparatively greater difference between the attacked and the original image.

Table 1: Average Image Norm Between Model Types

Model	**Dave-2**	**VGG**	**ResNet**
Average Image Norm	1684	1451	1973

Average Change in Steering Angle

This data point shows on average how much an adversarial attack changes the steering angle for each model. It is calculated by taking the steering angle from the frame before the attack (1/20th of a second before) and setting that as the expected angle. Then we subtract the adversarial steering angle from the expected angle and average overall frames with attacks. Between this value and the image norm, there is a pattern emerging where the ResNet model comparatively has higher values, followed by Dave-2. This means that these models are more susceptible to adversarial input and that adversarial examples cause a greater impact on their ability to drive the virtual vehicle.

Table 2: Average Change in Steering Angle Between Model Types

Model	**Dave-2**	**VGG**	**ResNet**
Average Change in Steering Angle	0.09401	0.0457	1.08224

Attack Success Rate

The attack success rate is a measure of how many attacks out of the total were able to change the expected steering angle by more than 5%. This rate was computed by taking the angle from the frame directly in front of an attack, setting it as the expected angle, and comparing it to the model's predicted angle on an adversarial image. The threshold of 5% was chosen as a measure of a successful attack because any less could potentially be attributed to variance within the machine learning model's training. Additionally, 5% is easily seen to have a large impact on the heading of the virtual vehicle. This data point furthers the emerging trend of adversarial susceptibility between the models.

Table 3: Attack Success Rates Between Model Types

Model	**Dave-2**	**VGG**	**ResNet**
Attack Success Rate	25.8%	11.7%	97.6%

Cross-Track Error

The vehicle's location was quantified by a variable returned by the simulator named cross-track error (CTE). The data used to train the models had a mean cross-track error of 0.132 with a standard deviation of 1.04. These values mean that there is a slight favor towards the right of the centerline throughout the data from the human expert, and there is a limited deviation from that position. Each run, the models with and without attacks also collected CTE values; these values will be compared in the next section. These results as a whole move towards answering the research question and finding if there is a difference in the impact of adversarial examples caused by model architecture.

Analysis

The hypothesis of model architecture having a significant impact on adversarial examples' efficacy is proven to be accurate through the data collected. Although there is no quantitative value to show the immediate efficacy of adversarial examples against different model architectures, we can achieve a conclusion by combining the data from the statistical tests and quantitative values. The first way to show a significant change in the cross-track error (CTE) was a statistical F-test between runs where the model was both under attack and not under attack. An F-test was chosen because each set of collected data will have a similar mean and only standard deviation/variance changes between each run by nature of the cross-track error. Before running this test, the collected CTE data distributions were compared to a normal distribution using a QQ plot to ensure the F-test can be run. All datasets met the conditions necessary for the statistical test. The F-test allows for accepting that the adversarial attack firstly causes a statistically significant difference on the cross-track error and secondly cements that cross-track error can be used as a method to measure the difference between the runs. This round of F-tests compares the CTE values between attacked and not attacked runs of each model. The p-value in this test shows the certainty in which the attack is causing a difference in the cross-track error. Since all of the p-values are under the threshold of 0.05, there is a statistically significant difference between the runs with and without an adversarial attack.

Table 4: F-Test between runs with and without adversarial attacks

Model	**Dave-2**	**VGG**	**ResNet**
F-value	1.041	1.089	1.109
P-value	0.02334	$1.544*10^{-5}$	$2.411*10^{-7}$

A second round of F-test is then run between the attacked runs of each model architecture to determine if there is a difference in CTE because of the change in model architecture. Since all p-values are less than 0.05 and the f-values are above 1 in each of the tests comparing the attacked runs from each model architecture, we can conclude that their model architecture causes a statistically significant difference in models' susceptibility to adversarial attacks.

Table 5: F-test between models with attacked runs

Comparison	**Dave-2 vs. VGG**	**Dave-2 vs. ResNet**	**VGG vs. ResNet**
F-value	1.550	1.755	2.720
p-value	$1.110x10^{-16}$	$1.110x10^{-16}$	$1.110x10^{-16}$

If we order the models in terms of how well they protect against attacks by using the differences proved by the second f-test and the average adversarial change in steering angle, we get VGG, Dave-2, and then ResNet. Essentially, the model based on the VGG architecture protects against adversarial attacks better than the other two model architectures and so on. This ordering of tested model architectures shows that in the experiment, shallower models with a greater number of filters on each layer are better protected against adversarial input.

Limitations

One limitation of my research is that this type of white-box attack is unlikely to happen in the real world. A real-world adversarial attack might only have access to the camera stream, for example. In contrast, in my research, the attacking algorithm had complete control over the camera stream, the model architecture, and the model weights to generate an adversarial image. Another limitation is that the tested models were end-to-end; in real-world systems, self-driving stacks are usually complex amalgamations of hardcoded algorithms and trained machine learning networks.

The adversarial examples used in this research were not created to be imperceptible to the human eye. For attacks in the real world, the adversarial images must not only evade detection from human supervisors but also adversarial image detection software. This means that adversarial examples against actual vehicles might have less of an impact on the steering angle; however, this cannot be known for sure.

Most attacks on software in the real world tend to bypass complicated machine learning systems by exploiting vulnerabilities in the code surrounding these systems [18]. This means that simply making a secure machine learning system is not sufficient; the code and software around the machine learning algorithms must also be incredibly secure. Although this research proves that there are potentially stronger model architectures to protect against adversarial input, there must still be sound software surrounding machine learning systems.

Implications

Although this research was focused on applications in self-driving, any situation where machine learning models can be applied should have a similar impact of model architecture on adversarial examples' effectiveness. This is due to the innate properties of machine learning models being able to generalize well to various tasks and applications [5]. Additionally, this study proves that there are ways of improving model architecture to better protect against adversarial examples. This property can apply to all applications of machine learning in that programmers and researchers can optimize their models to better protect against adversarial examples.

Although model architectures can inherently create problems in susceptibility to adversarial examples, they are usually kept constant when companies try to protect against adversarial examples. This is because model architectures have specific impacts on how the model behaves. Although this was minimally apparent in my research, changing model architecture to protect against adversarial examples is not always feasible or possible.

Overall, my research has shown that changing the architecture of end-to-end models changes the impact of adversarial examples in self-driving applications. However, since machine learning generalizes to many fields while keeping properties across them, architecture will most likely change the impact of adversarial examples in various other applications as well.

Next Steps

The primary direction for future research must be towards identifying which filters, layers, and structures impact a specific model's susceptibility to adversarial examples the most. This will allow researchers and programmers to build secure machine learning systems for many applications and ensure the safety of users and companies involved. Another area of future research is why specific structures cause increased susceptibility to adversarial input. If this is answered, we could be able to create the next generation of secure machine learning systems.

Conclusion

It has been proven that model architecture does have a statistically significant impact on the effectiveness of adversarial examples. This study begins the research needed to completely answer the question of how adversarial examples

are impacted by model architecture. Future research must look directly into what amount of filters, layers, and structures cause differences in the impact of adversarial examples. The implications for this discovery are far-reaching as companies and researchers not only need to design their models to accurately function on a given task but also design machine learning systems for security against malicious attacks.

Acknowledgements

I would like to thank Dr. Marianne Campbell for her help throughout the year and unending well of resources she offered during the research and writing process. I would also like to thank Dr. Vijay Gandapodi for all his assistance and guidance while planning and drafting.

References

[1] K. Hao, "What is machine learning?," MIT Technology Review, Nov. 17, 2018. https://www.technologyreview.com/2018/11/17/103781/what-is-machine-learning-we-drew-you-another-flowchart/ (accessed Sep. 08, 2020).

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," 2016, pp. 770–778. Accessed: Mar. 04, 2021. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html

[3] M. Bojarski et al., "End to End Learning for Self-Driving Cars," ArXiv160407316 Cs, Apr. 2016, Accessed: Sep. 11, 2020. [Online]. Available: http://arxiv.org/abs/1604.07316

[4] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The Limitations of Deep Learning in Adversarial Settings," in 2016 IEEE European Symposium on Security and Privacy (EuroS P), Mar. 2016, pp. 372–387. doi: 10.1109/EuroSP.2016.36.

[5] C. Szegedy et al., "Intriguing properties of neural networks," ArXiv13126199 Cs, Feb. 2014, Accessed: Sep. 28, 2020. [Online]. Available: http://arxiv.org/abs/1312.6199

[6] J. Canny, "A Computational Approach to Edge Detection," IEEE Trans. Pattern Anal. Mach. Intell., vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.

[7] T. Kanade, C. Thorpe, and W. Whittaker, "Autonomous land vehicle project at CMU," in Proceedings of the 1986 ACM fourteenth annual conference on Computer science - CSC '86, Cincinnati, Ohio, United States, 1986, pp. 71–80. doi: 10.1145/324634.325197.

[8] T.-D. Do, M.-T. Duong, Q.-V. Dang, and M.-H. Le, "Real-Time Self-Driving Car Navigation Using Deep Neural Network," in 2018 4th International Conference on Green Technology and Sustainable Development (GTSD), Nov. 2018, pp. 7–12. doi: 10.1109/GTSD.2018.8595590.

[9] PyTorch, PyTorch at Tesla - Andrej Karpathy, Tesla, (Nov. 06, 2019). Accessed: Mar. 26, 2021. [Online Video]. Available: https://www.youtube.com/watch?v=oBklltKXtDE

[10] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," Pattern Recognit., vol. 84, pp. 317–331, Dec. 2018, doi: 10.1016/j.patcog.2018.07.023.

[11] "Donkey® Car," Donkey® Car. https://www.donkeycar.com/ (accessed Mar. 04, 2021).

[12] A. Boloor, X. He, C. Gill, Y. Vorobeychik, and X. Zhang, "Simple Physical Adversarial Examples against End-to-End Autonomous Driving Models," in 2019 IEEE International Conference on Embedded Software and Systems (ICESS), Jun. 2019, pp. 1–7. doi: 10.1109/ICESS.2019.8782514.

[13] N. Patel, P. Krishnamurthy, S. Garg, and F. Khorrami, "Adaptive Adversarial Videos on Roadside Billboards: Dynamically Modifying Trajectories of Autonomous Vehicles," in 2019 IEEE/RSJ International Conference

on Intelligent Robots and Systems (IROS), Nov. 2019, pp. 5916–5921. doi: 10.1109/IROS40897.2019.8968267.

[14] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," ArXiv14091556 Cs, Apr. 2015, Accessed: Mar. 04, 2021. [Online]. Available: http://arxiv.org/abs/1409.1556

[15] S. Kundu, S. Prakash, H. Akrami, P. A. Beerel, and K. M. Chugg, "A Pre-defined Sparse Kernel Based Convolution for Deep CNNs," ArXiv191000724 Cs, Oct. 2019, Accessed: Mar. 16, 2021. [Online]. Available: http://arxiv.org/abs/1910.00724

[16] J. Jung, "Autonomous R/C Car Behavioral Cloning Optimization", [Online]. Available: http://cs229.stanford.edu/proj2018/report/51.pdf

[17] M. Uřičář, P. Křížek, D. Hurych, I. Sobh, S. Yogamani, and P. Denny, "Yes, we GAN: Applying adversarial techniques for autonomous driving," Electron. Imaging, vol. 2019, no. 15, pp. 48-1-48–17, Jan. 2019, doi: 10.2352/ISSN.2470-1173.2019.15.AVM-048.

[18] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," ArXiv14126572 Cs Stat, Mar. 2015, Accessed: Oct. 27, 2020. [Online]. Available: http://arxiv.org/abs/1412.6572

[19] B. Schneier, "Attacking Machine Learning Systems," Computer, vol. 53, no. 05, pp. 78–80, May 2020, doi: 10.1109/MC.2020.2980761.

Appendix A: Detailed Model Information

Training constants

```
#TRAINING
#The DEFAULT_MODEL_TYPE will choose which model will be created at training time. This chooses
#between different neural network designs. You can override this setting by passing the command
#line parameter --type to the python manage.py train and drive commands.
DEFAULT_MODEL_TYPE = 'dave2'    #(linear|categorical|rnn|imu|behavior|3d|localizer|latent|dave2|vgg|resnet)
BATCH_SIZE = 128                #how many records to use when doing one pass of gradient descent.
TRAIN_TEST_SPLIT = 0.8          #what percent of records to use for training. the remaining used for validation.
MAX_EPOCHS = 100                #how many times to visit all records of your data
SHOW_PLOT = True                #would you like to see a pop up display of final loss?
VERBOSE_TRAIN = True            #would you like to see a progress bar with text during training?
USE_EARLY_STOP = True           #would you like to stop the training if we see it's not improving fit?
EARLY_STOP_PATIENCE = 5         #how many epochs to wait before no improvement
MIN_DELTA = .0005               #early stop will want this much loss change before calling it improved.
PRINT_MODEL_SUMMARY = False     #print layers and weights to stdout
OPTIMIZER = 'adam'              #adam, sgd, rmsprop, etc.. None accepts default
LEARNING_RATE = 0.0001          #only used when OPTIMIZER specified
LEARNING_RATE_DECAY = 0.0       #only used when OPTIMIZER specified
SEND_BEST_MODEL_TO_PI = False   #change to true to automatically send best model during training
CACHE_IMAGES = True             #keep images in memory. will speed succesive epochs, but crater if not enough mem.
SEND_WANDB = False              #send values from training to weights and biases

PRUNE_CNN = False               #This will remove weights from your model. The primary goal is to increase performance.
PRUNE_PERCENT_TARGET = 75       # The desired percentage of pruning.
PRUNE_PERCENT_PER_ITERATION = 20 # Percenge of pruning that is perform per iteration.
PRUNE_VAL_LOSS_DEGRADATION_LIMIT = 0.2 # The max amout of validation loss that is permitted during pruning.
PRUNE_EVAL_PERCENT_OF_DATASET = .05  # percent of dataset used to perform evaluation of model.
```

Model Architectures

Dave-2 Model Architecture

Layer (type)	Output Shape	Param #
img_in (InputLayer)	[(None, 66, 200, 3)]	0

batch_normalization	(None, 66, 200, 3)	12
conv2d_1 (Conv2D)	(None, 31, 98, 24)	1824
dropout (Dropout)	(None, 31, 98, 24)	0
conv2d_2 (Conv2D)	(None, 14, 47, 36)	21636
dropout_1 (Dropout)	(None, 14, 47, 36)	0
conv2d_3 (Conv2D)	(None, 5, 22, 48)	43248
dropout_2 (Dropout)	(None, 5, 22, 48)	0
conv2d_4 (Conv2D)	(None, 3, 20, 64)	27712
dropout_3 (Dropout)	(None, 3, 20, 64)	0
conv2d_5 (Conv2D)	(None, 1, 18, 64)	36928
dropout_4 (Dropout)	(None, 1, 18, 64)	0
flattened (Flatten)	(None, 1152)	0
dense (Dense)	(None, 100)	115300
dropout_5 (Dropout)	(None, 100)	0
dense_1 (Dense)	(None, 50)	5050
dropout_6 (Dropout)	(None, 50)	0
angle_out (Dense)	(None, 1)	51

Total params: 251,761
Trainable params: 251,755
Non-trainable params: 6

VGG Model Architecture

Layer (type)	Output Shape	Param #
img_in (InputLayer)	[(None, 66, 200, 3)]	0
conv2d_1 (Conv2D)	(None, 66, 200, 32)	896

max_pooling2d	(None, 33, 100, 32)	0
conv2d_2 (Conv2D)	(None, 33, 100, 64)	18496
max_pooling2d_1	(None, 16, 50, 64)	0
conv2d_3 (Conv2D)	(None, 16, 50, 128)	73856
max_pooling2d_2	(None, 8, 25, 128)	0
conv2d_4 (Conv2D)	(None, 6, 23, 128)	147584
max_pooling2d_3	(None, 3, 11, 128)	0
dropout (Dropout)	(None, 3, 11, 128)	0
flattened (Flatten)	(None, 4224)	0
dense (Dense)	(None, 75)	316875
dropout_1 (Dropout)	(None, 75)	0
dense_1 (Dense)	(None, 50)	3800
dropout_2 (Dropout)	(None, 50)	0
angle_out (Dense)	(None, 1)	51

Total params: 561,558
Trainable params: 561,558
Non-trainable params: 0

ResNet Model Architecture

Excluded due to length (67 layers deep with residual passages increasing complexity)

Total params: 496,589
Trainable params: 494,823
Non-trainable params: 1,766

Full code can be accessed at https://bit.ly/3yltfCI