Department of Computer & Electrical Engineering and Computer Science
777 Glades Road, Boca Raton, FL 33431, USA

Adversarial Machine Learning

Christian Cruz & Ayman Moubayed
{cruzc2018, amoubayed2017} @fau.edu

Abstract. The growing myriad that is metadata, combined with the notable practical innovations over the last several years, have made machine learning into a crucial tool distributed across a wide lineup of fields including healthcare, automation, and economics. However, the achievement has been followed with many different important oppositions. In nature, many implementations of machine learning are subject to adversarial attacks. Some of the applications are security sensitive, such as a network for the exchange of payments or hospital records. In these applications, an adversary can be a mischievous group aimed at creating obstruction or faults within the model; to purposely play with the model and reveal some susceptibility within. In the vast world of technology that we live in today, to have a system with susceptibility to exploits can, for example, lead to possible security breaches and even important data loss. Some other applications might be adversarial because either its nature is or the data it is working with is. For example, a key set of problems in the field of security demand the spotting of spam and spyware. The use of machine learning in the given environment, creates a motivation among adversaries to avoid detection by altering their methods of attack or the objects that they create. The field of adversarial machine learning has surfaced to learn the many different susceptibilities of machine learning in an adversarial environment, in hopes to prevent future manipulation of the field by creating new approaches to make learning more secure and not as permitting for attacks. After studying many machine learning concepts and the many different neural networks within, as well as common use-cases of these in an adversarial environment, we offer a small look into one of the many possible applications that machine learning has not to mention, a look into an adversarial approach to the model as well. Our example model will mainly focus on decision-time attacks. Decision-time attacks take place when an adversary alters the nature of an occurrence seen by a trained model at the time of prediction in order to purposely either cause faults or take advantage of the model. The neural network will mainly focus on the ability to recognize different traffic signs.

Keywords *- Adversarial machine learning, manipulation, susceptibility, decision-time attacks.*

1 Introduction

Adversarial machine learning is now having an instance in big tech industries, such as IBM, Microsoft and Google [1,2,3]. Most of these companies have indicated, different from their dedication to adjusting their established software reliability, themselves to securing their machine learning systems. Gartner, a worldwide research and consulting firm for companies in many fields including IT, published an account on adversarial machine learning stating that, "Application leaders must anticipate and prepare to mitigate potential risks of data corruption, model theft, and adversarial samples" [4]. There are plentiful causes as to why big tech companies might already be ahead of their competitors in regard to securing their models. To name a couple companies that in the recent years invested heavily in machine learning: Amazon, Tesla, and Google. These companies, no matter how big, were faced with some level of adversarial strike [5,6,7]. Not to mention, organizations such as International Organization for Standardization (ISO), are in the process or have already developed guidelines for companies to base the security of their machine learning systems on as well as determining which companies have the most risk of adversarial attacks based on the past. [8]. ISO is a non-governmental organization trying to take part in machine learning standards. That being said, there are governmental organizations around the world creating guidelines as well that are recommended towards companies. For example, the European Union has tried to take action in ensuring that companies are constructing their machine learning models securely by releasing official specifications to determine reliability and trustability of the models [9]. Since machine learning is a new and growing field, most companies are looking to implement some kind of model of their own; however, because of how new and the remaining knowledge left to know in the field, companies are not developing the most secure systems and are looking for some kind of counsel.

1.1 Motivation and Contributions

Over the past couple decades, technology continues to grow, non-stop pushing to new heights. One of the many technological advances include machine learning; however, with the push to higher heights comes common security issues. Security issues such as adversarial machine learning. Earlier, it was established that both non-governmental and governmental organizations see the growth in the field of machine learning and have set forth some initial guidelines to follow with the attempt to increase security against adversaries. However noble these guidelines are, there will always be future discovered exploits. Our motivation with this paper is to help bring public awareness to the susceptibility that some neural networks have in hopes to generate applicable solutions to the field that will help to increase the security and reliability of future models to come. Our contributions consist of external research conducted on many neural networks as well as looking into some adversarial attacks on big name companies like Tesla [6]. To further prove how much more research is needed in the field before giving machine learning models our complete trust, we create and manipulate a convolutional neural network, written in Python, and discover some general faults common throughout many CNN's.

1.2 Organization of Article

The paper is organized as follows, section 2 is a comprehensive preliminary background on convolutional neural networks for novice readers. Section 3 aims to provide an example of an adversarial manipulation of a neural network in the field of automation. Section 4 is a technical discussion on the experimented upon neural network used in the previous section. Finally, section 5 will be closing remarks on adversarial machine learning as a whole versus our found neural network faults.

2 Preliminary Background

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other [10]. The main purpose of a convolutional neural network (CNN) is to try to perceive the world as we humans do. To understand and apply the learned knowledge to other fields such as image and sound recognition. Over time, CNNs have been developed into the sophisticated algorithm that it is and is even capable of tasks such as image tagging, visual search, and signal processing. This section will discuss the preliminary background on CNN that is needed to not only appreciate the sophisticated technology, but to also be able to recognize some possible loopholes.

2.1 CNN Architecture

The main architecture surrounding CNN is based on the basis that the input will be an image or a set of images. This forces the building of the neural network to be made in such a way that it is maximized to deal with the specific data set. One notable difference between CNN and other networks is that the neurons within a CNN are split into a three-dimensional structure, with each set of neurons analyzing a small region or feature of the image [11]; the spatial dimensionality of the input (height and the width) and the depth. Unlike traditional artificial neural networks, the depth does not mean the complete number of layers within, but rather the third dimension of an activation volume.

2.2 Overall Architecture

Convolutional neural networks are a particular type of neural networks that are normally composed of the following layers: convolutional layer, pooling, and fully connected layers [12]. When these layers are put together, one has a CNN. A visual representation of the layered network can be found in Figure 1.

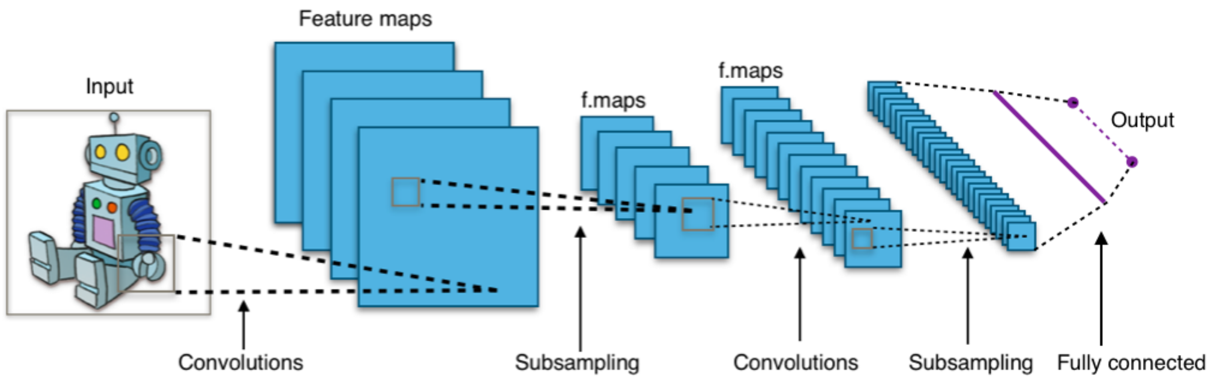


Fig. 1: Simplified image representing the layers of a CNN [25].

To further explain the presented image above, one can break down the image into four crucial regions.

1. Input layer: The input layer can consist of all the inputs for the CNN. In the specific neural network of image processing, the input layer is mostly composed of the pixel matrix of the desired image [13].
2. Convolutional layer: The convolutional layer is the main and probably most important part of the neural network. This layer has native connections and weights of common characteristics. The main point of the convolutional layer is to grasp characteristic representations of the inputted image [14].
3. Pooling layer: The pooling layer provides a go about for down sampling feature maps by recapping the existence of features in patches of the feature map. Two of the more common pooling methods are, average pooling and max pooling [15].
4. Fully Connected layer: The fully connected layer in a neural network are the layers where all the inputs from one layer are connected to every activation unit of the next layer. In some of the more general machine learning models, the last couple layers of the model are fully connected layers which compile the data set taken from previous layers to create the final output [16].

Through the layers discussed above, CNNs are able to analyze the original input layer using convolutional and downsampling approaches to create class scores for classification and regression purposes. However, it is crucial to note that this is just the general structure for a CNN. For example, different models might be less linear and have more activation functions or have used the dropout technique. Dropout is used when a model becomes too dependent on the dataset (or overfitted) and when it is time to predict, will produce inaccurate results.[17]. To fix this, neuron results or outputs will be dropped out which will produce results that are not overfitted to the dataset. It takes time to not only create these models but to also develop a deep enough understanding of them to be able apply a model to a specific field. The rest of this section will be spent further reviewing the specified layers above.

2.3 Convolutional Layer

This layer has a very important role to play in CNNs. The parameter set for this layer is centered around the use of lernable kernels. These kernels are commonly small in terms of dimensions, but can go along the entire depth of the dimensioned data from the inputted image. When the data gets to the convolutional layer, the layer convolves each filter/kernel along the entire dimensionality of the data from the inputted image, in order to create 2D activation maps or feature maps. As the process continues, a scalar value is calculated as a result from using the input data set and a kernel/filter. At this point, the model is able to study the scalar values produced and relate them to specific characteristics [18]. These values will then become known as activations. The image below can help give a visual representation of the recently discussed process:

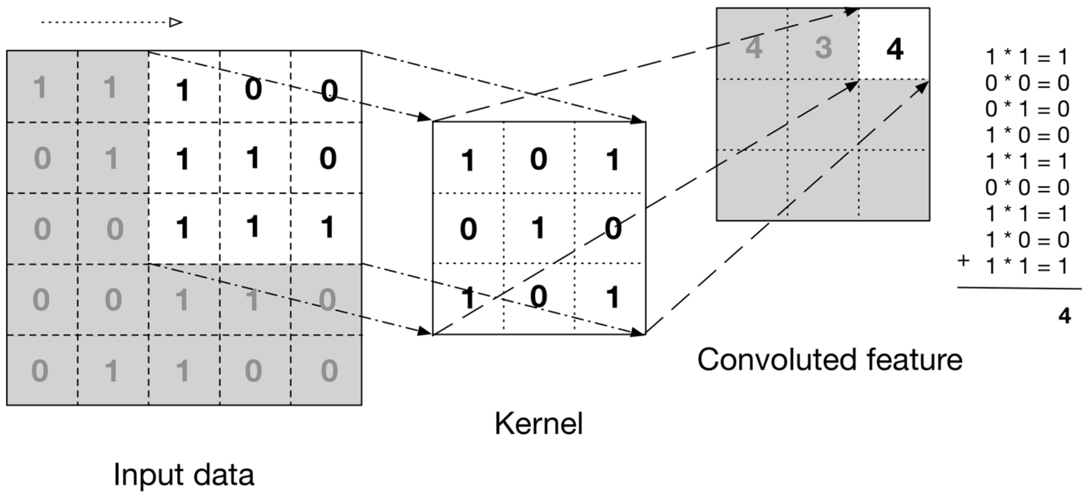


Fig. 2 Visual representation of the convolutional layer [26].

In a model, the use of different kernels/filters can be used for specific operations such as edge detection, blur, and sharpening [19]. Depending on what the model is being used for, such operations can be extremely effective in a field.

2.4 Pooling Layer

The main point of the pooling layer is to try and reduce the dimensionality of the representation. In doing so, it not only reduces the difficulty of the model but reduces the number of parameters as well. In a typical CNN, a possible attempt at trying to lower the dimensionality comes in the form of a max-pooling layer with the 2 x 2 dimensions applied with a stride of 2 across the dimensionality of the input [20]. In doing so, the pooling layer is able to cut down the size by 25% of the original size; however, even though the size was cut down by 25%, the depth remains untouched and stays the same. Similar to max pooling is min pooling, essentially the same as max pooling but instead of taking the maximum, one takes the minimum instead. Another possible pooling method is average pooling, where instead of taking the minimum or maximum, one takes the average of the corresponding block.

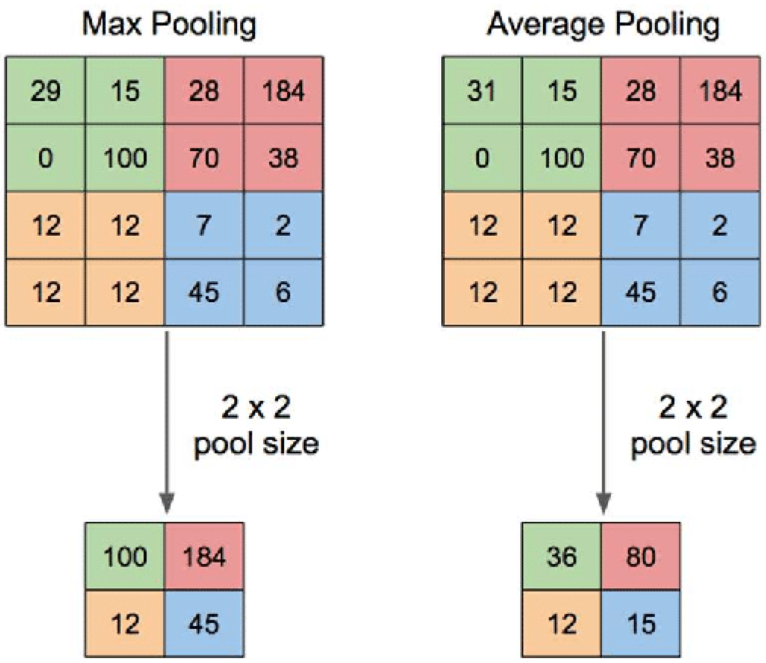


Fig. 4: Visual representation of the max and average pooling method [27].

2.5 *Fully Connected Layer*

Fully connected layer is full of neurons that are directly connected to the neurons in the two nearby layers. It is also worth mentioning that they are not connecting to any of the neurons within itself, only to the two nearby layers. To visualize this more, it is a similar set up as a traditional artificial neural network.

2.6 *Activation Functions*

For a neural network, activation functions are a crucial part of the model. Generally, activation functions and bais go hand and hand. Using a bias can give one the ability to shift the activation function by adding a constant [21], which oftentimes is what can make the model more accurate. It can be said that the activation function determines a neural network's linearity. Below are some of the more common activation functions:

1. Sigmoid Functions - Classified as a nonlinear function, sigmoid functions are generally used in a feedforward network where the connection between the nodes do not cycle.
2. Hyperbolic Tangent Function (Tanh) - Classified as a nonlinear function and more commonly used in multilayer neural networks as opposed to a sigmoid function because tanh provides better training performance.
3. Softmax Function - Classified as a nonlinear function and used in neural networks to calculate the probability distribution.
4. Softsign Function - Classified as a nonlinear function and most commonly used in regression computation problems. Unlike tanh that covers exponentially, softsign covers polynomials.
5. Rectified Linear Unit (ReLU) Function - Classified as a nonlinear function and is the most popular activation function used because of its speed and generalization in deep learning.
6. Exponential Linear Units (ELUs) Function - Classified as a nonlinear function and used to speed up the training of the model. One advantage to ELU is that it can get rid of the vanishing gradient problem by using identification for positive results.

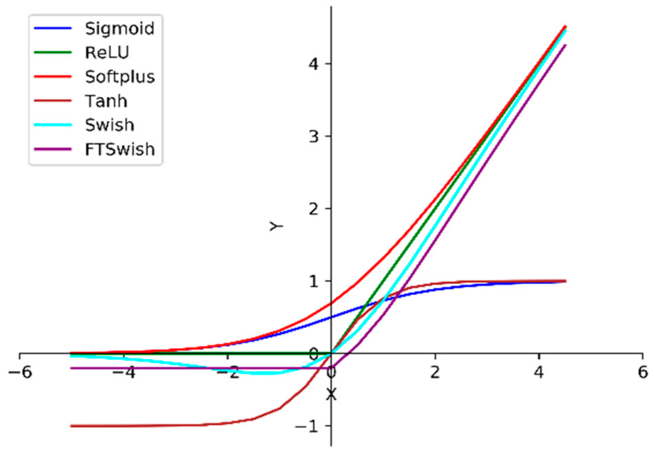


Fig. 5: Visual representation of different activation functions [28].

3 Research

Over the years, neural networks have been becoming more relevant in the field of automation. Their decision-making potentiality can be used almost anywhere in the field to make systems more efficient and secure. Big name companies such as Tesla, have already incorporated tons of neural networks into their very well known autopilot AI [22]. Their autopilot AI applies top of the line research to train neural networks on some common issues ranging from perception to control. From the many cameras around the vehicle, it is able to detect cars, people, traffic signs, and crosswalks. For this section and as an example of the impact CNN's can have in a field such as automation in terms of vehicle transport, we will cover a network that we developed to detect traffic indicators and show the importance of having security against adversarial attacks. It is important to note that the data set consists of german road signs and the network was built using Python with libraries such as: tensorflow, keras, matplotlib, numpy, pandas, skimage.

3.1 Overall CNN Architecture

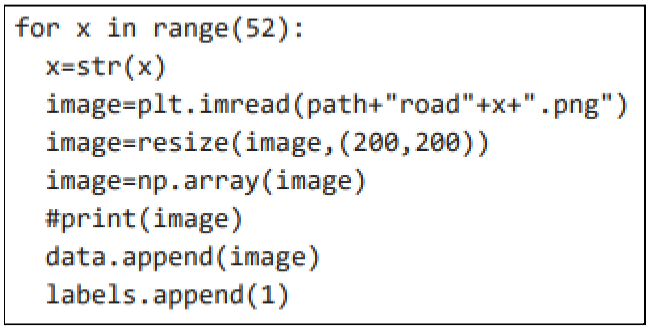
The network was built using several different python libraries that made the design and requirement process easier. To start, the network is sampling from a source of 160 different images that are categorized as follows: traffic lights, stop sign, speeding limit, crosswalks. To start, the images are converted into their corresponding matrix and then fed into the convolutional layer. In the convolutional layer, the dataset then goes through three activation functions before finally determining the categorical cross entropy. The following sections below will further explain the different sections of the network.

3.2 Dataset

Our dataset comes from a very popular online community that is centered around data science and machine learning. The Kaggel website provides many large datasets available to the community for educational and professional purposes. The website did not contain any traffic indication images in the United states, but did contain a dataset of traffic indication images in Germany. We were able to download the dataset that contained 877 high quality images but only used about 160 in order to store and process our images more effectively. Our accuracy using only 160 images for training and testing was fair and with a greater dataset would most likely increase the accuracy of our network.

3.3 Pre-Processing

As part of the pre-processing phase, a certain percentage of the dataset was allocated for testing while another percentage was allocated for training. It is said to be a rule of thumb that about 20% of the dataset is meant to be allocated for testing but because the dataset used for our network was small (160 images), only 10% of the dataset was allocated for testing while 90% was allocated for training. For this network, the images were kept in a Google drive which is a free cloud base storing platform with limited capacity. Therefore, we had to connect to the cloud storing platform and then fetch each individual image. Because there are four main sections the images could fall under (traffic light, stop sign, speed limit, crosswalk), the dataset was queried in sections. Images 0-51 contained traffic lights, 52-99 contained stop signs, 100-121 contained speed limit signs, and 121-159 contained crosswalks. Each image was fetched from the drive and then stored in a list using metaplotlib's imread(). Then every image was resized to a width and a height of 200 pixels (200x200). The data was then converted into a numpy array and appended to a list. The below image shows sample code for images 0-51. It is also important to note that the image classification was taken into a separate list (what type of image it was: traffic light, stop sign, speed limit, crosswalk).



```
for x in range(52):
  x=str(x)
  image=plt.imread(path+"road"+x+".png")
  image=resize(image,(200,200))
  image=np.array(image)
  #print(image)
  data.append(image)
  labels.append(1)
```

Fig. 6: Sample code taken from the network representing images 0-51.

3.4 Convolutional layer

This layer was composed of one activation function. The activation function was made of Relu and used in a 3 X 3 kernel. As stated earlier in section 2, Relu is a nonlinear activation function used in the form of max(0,x). Let the image below serve as a reminder:

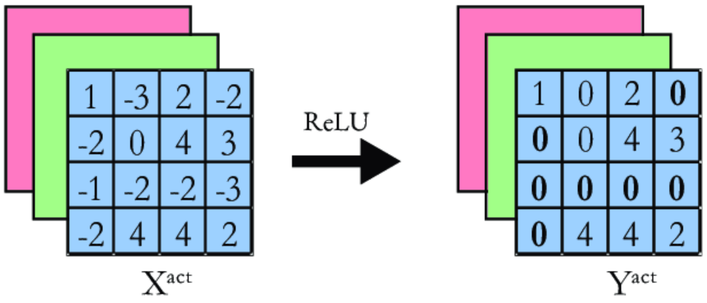


Fig.7. Visual representation of ReLu [29].

3.5 Adversarial Attack

Not long ago, neural networks have been achieving top of the line performance over several pattern recognition tasks, most recently visual classification problems. Convolutional neural networks are now able to identify objects in images with human-like execution; however, changing an image in a way that makes it unrecognizable to humans can cause CNNs to identify the image as something entirely different. It is just as easy to produce an image that is completely unrecognizable or to slightly alter an image/object so that one questions what it is but, some CNN's still proceed to identify the image/object with near absolute certainty. "Fooling" a network is when something that is slightly unrecognizable is identified by the network as something that it is not. A famous example was when a black sticker was placed on a stop sign and many high end networks in the automation field were no longer able to confirm if the image was a stop sign. The following section and experiment will present the results obtained from attempting an adversarial attack on our own baseline model.

3.6 Adversarial Attempt

As stated earlier, the dataset consisted of a total of 160 images of german road signs that were obtained from a very popular online community that is centered around data science and machine learning, the Kaggel website. Our developed network showed an accuracy of 75% after training the model and using the allocated 10% of the dataset as test images. With that in mind, a total of five images were used to try and fool the network; two were stop signs, two were speed limits, and one was a crosswalk sign. A sample image of the signs is shown below:

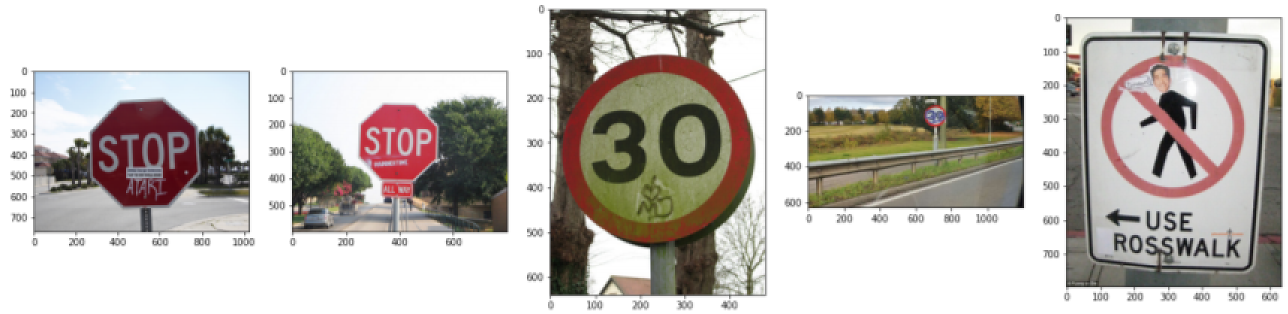
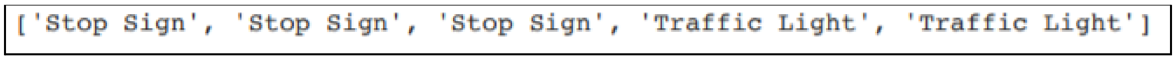


Fig. 8: Visual representation of the 'fooling images' used.

All five images consisted of some sort of attempt at trying to fool the network, whether it was by graffiti or just placing stickers on the signs. After running these images against the network, it was able to correctly recognize two out five; the first two images of stop signs. Below are the results of the network:



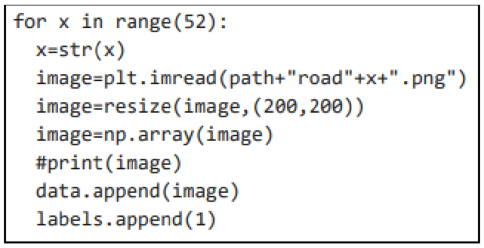
```
['Stop Sign', 'Stop Sign', 'Stop Sign', 'Traffic Light', 'Traffic Light']
```

Fig. 9: Visual representation of the results obtained from the adversarial attempt.

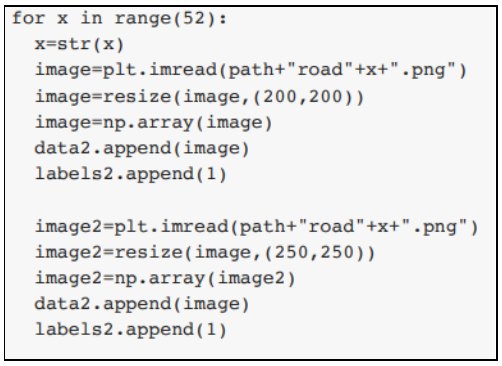
3.7 Proposed Solution

The importance of securing a network against adversarial attacks have been stressed through this entire paper through providing general background information, providing an example network applicable in the field of automation in regards to self driving vehicles, and providing an adversarial attempt on that same network. Our proposed solution was to increase the networks accuracy in hopes of lowering the amount of fooling. Below are the proposed changes:

1. Increase dataset - As mentioned earlier, when the images were being queried from Google Drive, a free cloud base storing platform with limited capacity, the images were first resized to 200x200 then appended to a list. To help increase the dataset, in addition to adding the original 200x200, the same image but in a 250x250 format was added. Essentially doubling the dataset. Below shows the difference between building the first and second model's dataset.



```
for x in range(52):
  x=str(x)
  image=plt.imread(path+"road"+x+".png")
  image=resize(image,(200,200))
  image=np.array(image)
  #print(image)
  data.append(image)
  labels.append(1)
```



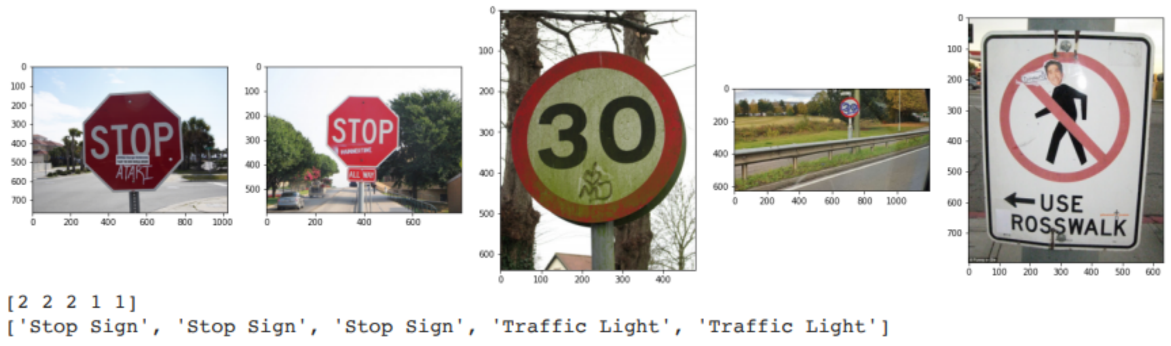
```
for x in range(52):
  x=str(x)
  image=plt.imread(path+"road"+x+".png")
  image=resize(image,(200,200))
  image=np.array(image)
  data2.append(image)
  labels2.append(1)

  image2=plt.imread(path+"road"+x+".png")
  image2=resize(image,(250,250))
  image2=np.array(image2)
  data2.append(image)
  labels2.append(1)
```

2. Validation - in the original network, validation was set at 10%; meaning that 10% of the data was allocated for testing purposes while 90% was for training. With the doubling of images in the dataset, the validation has been raised to 20%.
3. Additional layers - additional layers including dropout and average pooling have been added.
4. Epoch - in the original model, the epoch was at 10. In the changed model it is now 15.
5. Batch size - in the original model batch size was 6. In the new model the batch size is 32. Allowing more training per iteration.

3.8 Proposed Solution - Results

After the proposed solution was implemented, we saw an increase of accuracy from 75% to 92%. Between the doubling the data, adding more layers, and increasing the epoch, the model was able to increase the accuracy by 17%. With the increase in accuracy, we hoped to get better fooling results; however, the results came out the same as the original model. Below is the result of the model following the changes:

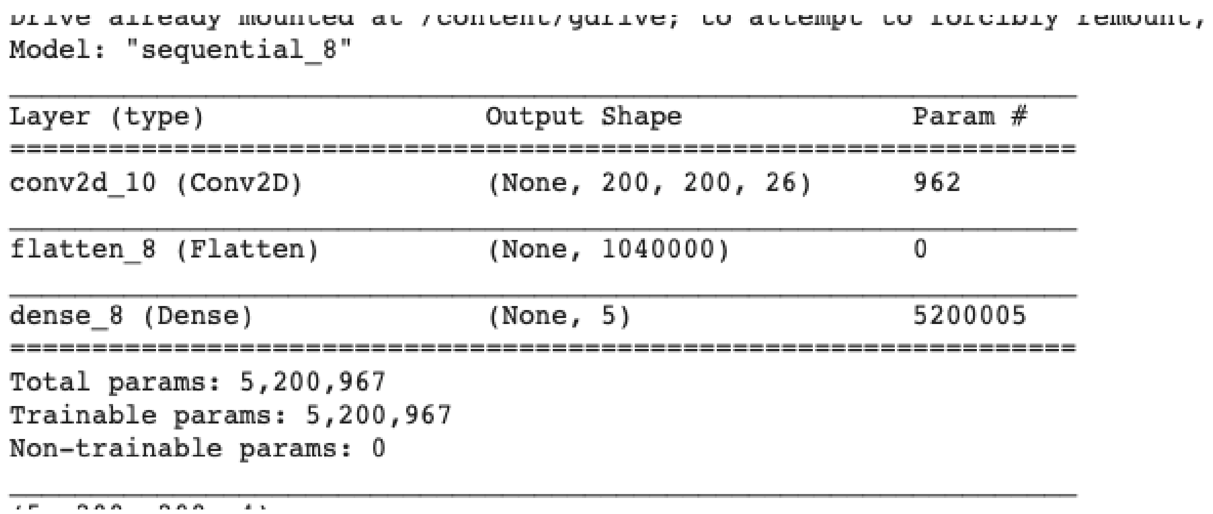


```
[2 2 2 1 1]
['Stop Sign', 'Stop Sign', 'Stop Sign', 'Traffic Light', 'Traffic Light']
```

Fig. 9. Visual representation of the results acquired from trying to fool the improved model. Even after increasing the accuracy by 17%, the model still managed to return the same results. Arguably, some of the images used for fooling were pretty severe.

4 Technical Discussion

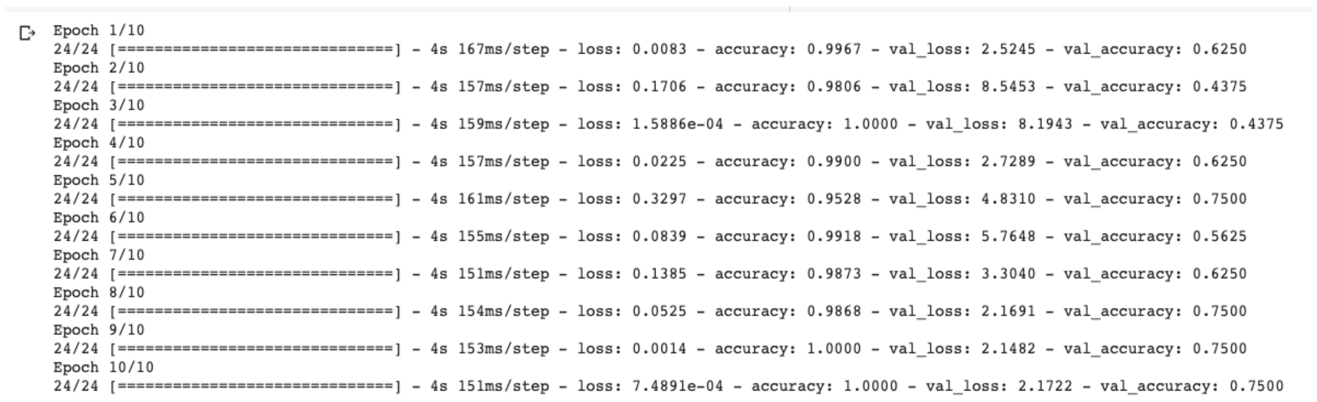
We began this project by trying to establish a simple convolutional neural network that would be able to classify common road images to create a baseline of what neural networks in automation may encounter with the hope of improving it in order to fight against adversarial attacks. The first convolutional neural network developed was created with only a single convolutional input layer since the dataset was too small to have any signifcates or ability to increase accuracy with more layers in the network. The architecture of the baseline convolutional neural network can be seen down below:



```
Drive already mounted at /content/gdrive; to attempt to forcibly remount,
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 200, 200, 26)      962
_________________________________________________________________
flatten_8 (Flatten)          (None, 1040000)           0
_________________________________________________________________
dense_8 (Dense)              (None, 5)                 5200005
=================================================================
Total params: 5,200,967
Trainable params: 5,200,967
Non-trainable params: 0
_________________________________________________________________
```

Fig. 10. Visual representation of the baseline model architecture.

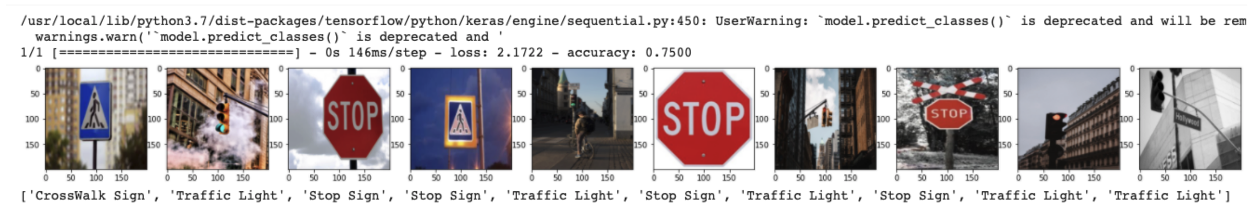
As a result of our small dataset passing through the baseline neural network, the accuracy and validation accuracy did not mirror one another as they should; rather, the baseline model radically flucculted in accuracy and the validation loss decreased too quickly as well as seen below.



```
Epoch 1/10
24/24 [==============================] - 4s 167ms/step - loss: 0.0083 - accuracy: 0.9967 - val_loss: 2.5245 - val_accuracy: 0.6250
Epoch 2/10
24/24 [==============================] - 4s 157ms/step - loss: 0.1706 - accuracy: 0.9806 - val_loss: 8.5453 - val_accuracy: 0.4375
Epoch 3/10
24/24 [==============================] - 4s 159ms/step - loss: 1.5886e-04 - accuracy: 1.0000 - val_loss: 8.1943 - val_accuracy: 0.4375
Epoch 4/10
24/24 [==============================] - 4s 157ms/step - loss: 0.0225 - accuracy: 0.9900 - val_loss: 2.7289 - val_accuracy: 0.6250
Epoch 5/10
24/24 [==============================] - 4s 161ms/step - loss: 0.3297 - accuracy: 0.9528 - val_loss: 4.8310 - val_accuracy: 0.7500
Epoch 6/10
24/24 [==============================] - 4s 155ms/step - loss: 0.0839 - accuracy: 0.9918 - val_loss: 5.7648 - val_accuracy: 0.5625
Epoch 7/10
24/24 [==============================] - 4s 151ms/step - loss: 0.1385 - accuracy: 0.9873 - val_loss: 3.3040 - val_accuracy: 0.6250
Epoch 8/10
24/24 [==============================] - 4s 154ms/step - loss: 0.0525 - accuracy: 0.9868 - val_loss: 2.1691 - val_accuracy: 0.7500
Epoch 9/10
24/24 [==============================] - 4s 153ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 2.1482 - val_accuracy: 0.7500
Epoch 10/10
24/24 [==============================] - 4s 151ms/step - loss: 7.4891e-04 - accuracy: 1.0000 - val_loss: 2.1722 - val_accuracy: 0.7500
```

Fig. 11. Visual representation of the baseline model training.

After training the baseline neural network, we tested it on some sample images from the test set to observe its accuracy. When presented with 10 random images the 75% accurate baseline network was able to score a 9/10 in our test images as shown below.

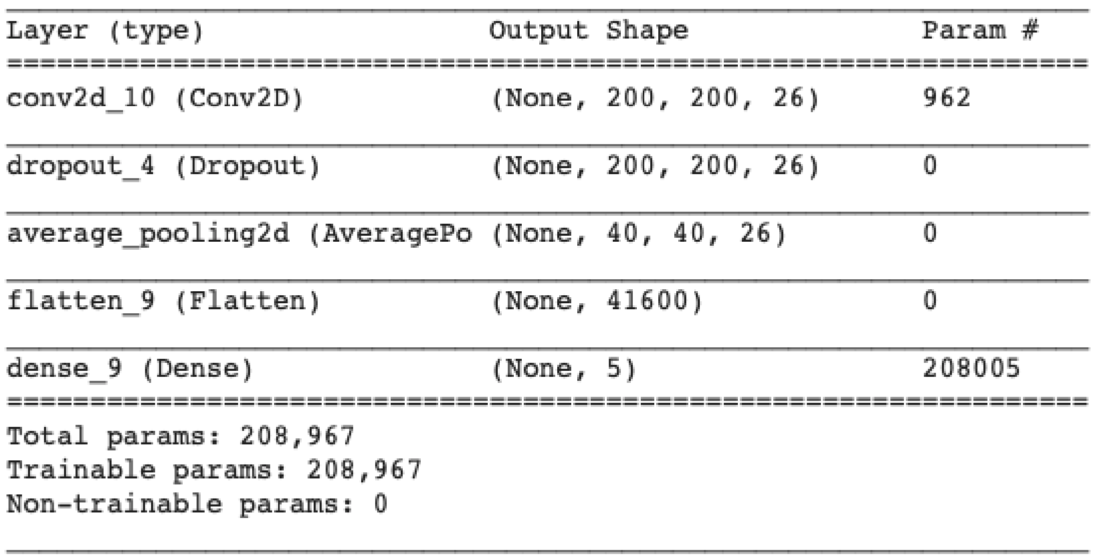


```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning: `model.predict_classes()` is deprecated and will be rem
  warnings.warn('`model.predict_classes()` is deprecated and '
1/1 [==============================] - 0s 146ms/step - loss: 2.1722 - accuracy: 0.7500
['CrossWalk Sign', 'Traffic Light', 'Stop Sign', 'Stop Sign', 'Traffic Light', 'Stop Sign', 'Traffic Light', 'Stop Sign', 'Traffic Light', 'Traffic Light']
```

Fig. 12. Visual representation of the baseline model results.

However as shown earlier (figure 9), when presented with the fooling dataset, it performed poorly by only scoring 2/5 correctly. So based on the fooling result, we concluded that if we were to articulate a better neural network we would be able to score higher in our fooling test set. The main metrics to determine if the new network is better than the baseline network would be to compare the accuracy, loss, and training sessions. The first step to accomplish this would be to have more flexibility in the networks architecture. We were not able to add new networks or adjust the parameters in training because of the small dataset we possessed. Although there was more data available in the dataset we had a space and restival issue where the cloud base storage service we were using would be able to store more data without a payment transaction. With this limitation we came up with a creative solution that is very popularly used when more data does not exist, especially in the medical field. We can take data that we already possess and alternate it in size, orientation or filters and feeded it back to the network as new data. Although a human can tell that two images are the same but different size, the neural network would take the image in as a new image entry and would learn different characteristics from the two images. Therefore as shown in the proposed solution section, we simply took the same image and created new data by resizing it as 250x250 and adding it as another data entry therefore doubling our dataset. With more training we were able to acquire better weights for the network as a whole. Having a larger dataset, we are now finally able to add more layers to our baseline network and adjust some training parameters as we see fit. We began by changing the batch size to 32 and the epochs to 15 so that more images were being trained at a time to speed up the training of the network as we had many images and were training it slowly as if we had only a few to train from With our larger dataset we actually began having the opposite problem as before. Our baseline network was too simple and was becoming overfitted by the dataset. The solution to this of course, was to first implement a dropout layer. This seemed like the appropriate approach when a network is overfitted or too well trained to the training set. We want the model to be as nonlinear as possible in order to be applicable to general testing not just our training set. So therefore, by adding a dropout layer of 25% we can stop the network from being overfitted and learn a fraction of the weights during training[23]. Although we could offset it to 50% drop out for maximum regulation, we decided to only keep it at 25% in order to keep our accuracy high and still be able to introduce another layer to the network.

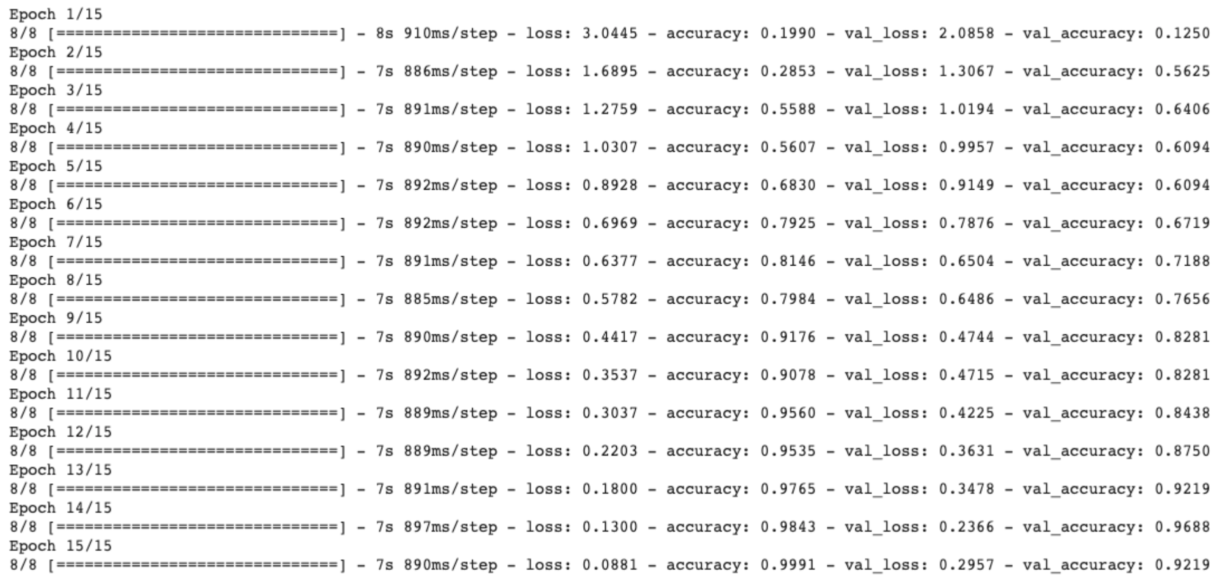
Most convolutional neural networks contain some kind of pooling layer. A max pooling layer would have been a good choice if we had more subtle and darker background images for our training set, however it is generally said that for images with crispr and more colourful backgrounds an average pooling layer may be better[24]. With our dataset containing lots of different bright colors for traffic lights and signs, it seemed like average pooling was a better choice in our secinaro rather than max pooling. The new convolutional neural network architecture can be seen below:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 200, 200, 26)      962
_________________________________________________________________
dropout_4 (Dropout)          (None, 200, 200, 26)      0
_________________________________________________________________
average_pooling2d (AveragePo (None, 40, 40, 26)        0
_________________________________________________________________
flatten_9 (Flatten)          (None, 41600)             0
_________________________________________________________________
dense_9 (Dense)              (None, 5)                 208005
=================================================================
Total params: 208,967
Trainable params: 208,967
Non-trainable params: 0
_________________________________________________________________
```

Fig. 14. Visual representation of the improved network architecture.

When we trained our newly improved neural network, we independently observed a big difference in training patterns. The new neural network trained in a much better manner than the baseline model. The accuracy and validation accuracy mirrored one another to a certain extent which shows that the network is learning correctly throughout the training process rather than fluctuating its correctness. There also is a natural growth in accuracy and decrease in loss as the neural network is being trained. The baseline network seemed somewhat unpredictable and fluenating as the new neural network seems more naturally stable and learning at a linear rate which is an observable improvement as seen below.



```
Epoch 1/15
8/8 [==============================] - 8s 910ms/step - loss: 3.0445 - accuracy: 0.1990 - val_loss: 2.0858 - val_accuracy: 0.1250
Epoch 2/15
8/8 [==============================] - 7s 886ms/step - loss: 1.6895 - accuracy: 0.2853 - val_loss: 1.3067 - val_accuracy: 0.5625
Epoch 3/15
8/8 [==============================] - 7s 891ms/step - loss: 1.2759 - accuracy: 0.5588 - val_loss: 1.0194 - val_accuracy: 0.6406
Epoch 4/15
8/8 [==============================] - 7s 890ms/step - loss: 1.0307 - accuracy: 0.5607 - val_loss: 0.9957 - val_accuracy: 0.6094
Epoch 5/15
8/8 [==============================] - 7s 892ms/step - loss: 0.8928 - accuracy: 0.6830 - val_loss: 0.9149 - val_accuracy: 0.6094
Epoch 6/15
8/8 [==============================] - 7s 892ms/step - loss: 0.6969 - accuracy: 0.7925 - val_loss: 0.7876 - val_accuracy: 0.6719
Epoch 7/15
8/8 [==============================] - 7s 891ms/step - loss: 0.6377 - accuracy: 0.8146 - val_loss: 0.6504 - val_accuracy: 0.7188
Epoch 8/15
8/8 [==============================] - 7s 885ms/step - loss: 0.5782 - accuracy: 0.7984 - val_loss: 0.6486 - val_accuracy: 0.7656
Epoch 9/15
8/8 [==============================] - 7s 890ms/step - loss: 0.4417 - accuracy: 0.9176 - val_loss: 0.4744 - val_accuracy: 0.8281
Epoch 10/15
8/8 [==============================] - 7s 892ms/step - loss: 0.3537 - accuracy: 0.9078 - val_loss: 0.4715 - val_accuracy: 0.8281
Epoch 11/15
8/8 [==============================] - 7s 889ms/step - loss: 0.3037 - accuracy: 0.9560 - val_loss: 0.4225 - val_accuracy: 0.8438
Epoch 12/15
8/8 [==============================] - 7s 889ms/step - loss: 0.2203 - accuracy: 0.9535 - val_loss: 0.3631 - val_accuracy: 0.8750
Epoch 13/15
8/8 [==============================] - 7s 891ms/step - loss: 0.1800 - accuracy: 0.9765 - val_loss: 0.3478 - val_accuracy: 0.9219
Epoch 14/15
8/8 [==============================] - 7s 897ms/step - loss: 0.1300 - accuracy: 0.9843 - val_loss: 0.2366 - val_accuracy: 0.9688
Epoch 15/15
8/8 [==============================] - 7s 890ms/step - loss: 0.0881 - accuracy: 0.9991 - val_loss: 0.2957 - val_accuracy: 0.9219
```

Fig. 15. Visual representation of the new model training.

The new neural network had an accuracy of 92% and a 8% validation loss which is a higher accuracy of about 17% and more stable then the baseline network. The grow can be mapped out as seen below.

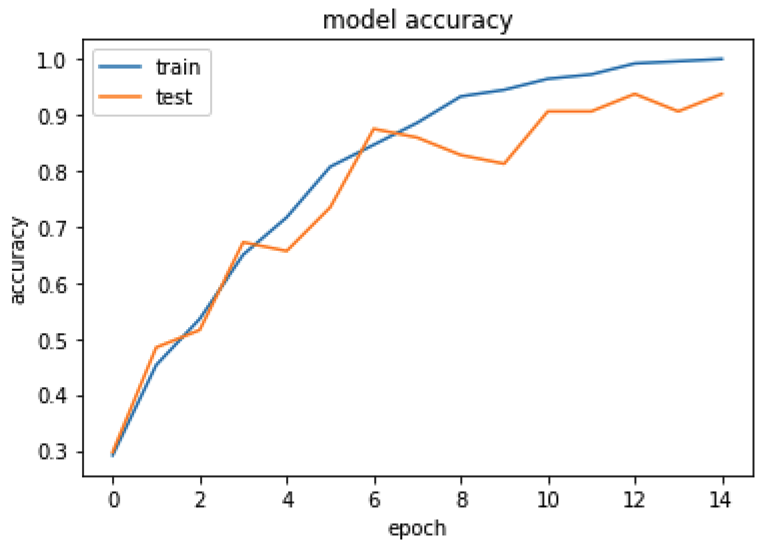


Fig. 16. Visual representation of the new model accuracy.

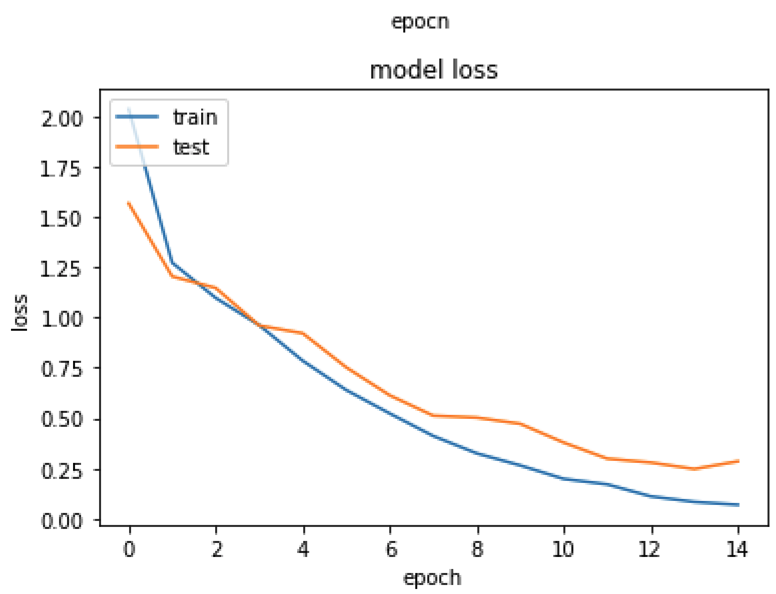


Fig. 17. Visual representation of the new model loss.

Although our new model was dramatically improved with a higher accuracy rate and a significantly better training, the new network still performed about the same when given the fooling image test set. Although our efforts were to improve the model in order to better deal with adversarial attacks such as fooling images, we were unable to improve our score with the fooling image set as talked about earlier. This is because, although our network is better prepared to classify these images at a higher accuracy, it is still unable to prevent or look past certain key characters in attacks that will be further talked about in the next section.

5 Concluding Remarks and Future Direction

Over the years, neural networks have been becoming more relevant in the field of automation. Their decision-making potentiality can be used almost anywhere in the field to make systems more efficient and secure. As stated earlier, big name companies such as Tesla, have already incorporated tons of neural networks into their very well known autopilot AI [22]. Their autopilot AI applies top of the line research to train neural networks on some common issues ranging from perception to control. From the many cameras around the vehicle, it is able to detect cars, people, traffic signs, and crosswalks. The importance of having not only an accurate model but a secure model, especially in regards to a self driving car, are crucial in keeping not only the drivers safe but everyone else as well. An adversarial party could attempt to fool several traffic signs in hopes of wreaking havoc. For example, take the speed limit image used in the fooling attempt (Fig 3., located in section 3.5). The network recognized the road sign as a stop sign. What would that mean in regards to the self driving car? If the sign was identified as a stop sign, that means the car would slow to a stop in the middle of a potentially busy intersection which could possibly lead to an accident happening. Affecting not only the life of the driver, but the lives of the other drivers around as well. This accident was caused by an adversarial party looking to exploit the network's vulnerability. This was just one example of a potential outcome of having an unsecure system. There could be many more different scenarios that could be much worse.

In our network development we observed that even if a network is built better to a higher accuracy, adversarial attacks will still occur. This is because at some point even a human will be confused and no longer able to interpret certain signs. The most challenging image shown in our fooling test set was a crosswalk sign that had some of the letters removed and a solid red cross through the image. By human perception, one may think that sign indicated no one should cross or jay walk in this area even though it was originally just a plain crosswalk sign. The reality of it is, after a certain amount of tampering, it is no longer feasible for a computer or human to be able to differentiate what the traffic sign represents. When the two stop signs in the fooling dataset contained a small amount of graffiti, the network was able to still look past the tampering and classify them as stop signs. Therefore one could conclude that the amount of tampering done by the adversary is what determines when the network will be fooled or not. If a road sign is tempered with beyond human recognition, it does not seem like a realistic task for a network to be able to still be able to detect it as of right now. As time progresses, networks will start to learn certain adversal attack characteristics such as the famous black tape on the stop sign example or simple graffiti. This will however require the network to have a basic understanding as the network we created but on a larger scale, and then begin training on adverisal attack attributes. In order for the network to learn those attributes, a separate research study would have to be conducted in order to determine the most common adversarial attack in specially road signs, and then find a large data set to train a separate network on and combine the two. With the right resources, a project like this certainly can be accomplished but would require lots of dedication and resources to fight against adversarial attack across all field machine learning takes place in.

References

[1] "Adversarial Machine Learning", Jul 2016, [online] Available: https://ibm.co/36fhajg.
[2] "Securing the Future of AI and ML at Microsoft", [online] Available: https://docs.microsoft.com/en-us/security/securing-artificial-intelligence-machine-learning.
[3] "Responsible AI Practices", [online] Available: https://ai.google/responsibilities/responsible-ai-practices/?category=security.
[4] S. A. Gartner, "Anticipate Data Manipulation Security Risks to AI Pipelines", [online] Available: https://www.gartner.com/doc/3899783.
[5] J. Li, S. Qu and X. Li, "Adversarial Music: Real World Audio Adversary Against Wake-word Detection System", Advances in Neural Information Processing Systems, pp. 11908-11918, 2019.
[6] Tencent Keen Security Lab. "Experimental Security Research of Tesla Autopilot", [online] Available: https://keenlab.tencent.com/en/whitepapers/Experimental_Security_Research_of_Tesla_Autopilot.pdf.
[7] A. Athalye, L. Engstrom, A. Ilyas and K. Kwok, "Synthesizing robust adversarial examples", arXiv preprint, 2017.
[8] R. Von Solms, "Information security management: why standards are important", Information Management & Computer Security, vol. 7, no. 1, pp. 50-58, 1999.
[9] "Ethics guidelines for trustworthy ai", Nov 2019, [online] Available: https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai.
[10] Sumit Saha. “A comprehensive Guide to Convolutional Neural Networks”, [online] Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2 B1164a53
[11] “Convolutional Neural Network Architecture: Forging Pathways to the Future”, [online] Available: https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-architecture-forgin g-pathways-future/
[12] Afshne Amidi and Shervine Amidi, “Convolutional Neural Network Cheat Sheet”, [online] Available: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks
[13] Xin Zhang, Yongcheng Wang, Ning Zhang, Dongdong Xu and Bo Chen, “Research on Scene Classification Method of High-Resolution Remote Sensing Images Based on RFPNet”, [online] Available: https://www.researchgate.net/publication/333159107_Research_on_Scene_Classification_Method_of_High -Resolution_Remote_Sensing_Images_Based_on_RFPNet
[14] Tianmei Guo, Jiwen Dong, Henjian Li and Yunxing Gao, “Simple convolutional neural network on image classification”, [online] Available: “https://ieeexplore.ieee.org/abstract/document/8078730”.
[15] Jason Brownlee, “A Gentle Introduction to Pooling Layers for Convolutional Neural Networks”, [online] Available: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/
[16] “Fully Connected Layer: The brute force layer of a Machine Learning model”, [online] Available: https://iq.opengenus.org/fully-connected-layer/
[17] “What is dropout in neural networks?”, [online] Available: https://www.educative.io/edpresso/what-is-dropout-in-neural-networks
[18] Géron, Aurélien, “Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow”, [media] Available: ISBN ISBN 978-1-492-03264-9., pp. 448
[19] Prabhu, “Understanding of Convolutional Neural Network (CNN) — Deep Learning”, [online] Available: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99 760835f148
[20] Dan Ciresan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, Jurgen Schmidhuber, “Flexible, High Performance Convolutional Neural Networks for Image Classification”, [online] Available: https://people.idsia.ch//~juergen/ijcai2011.pdf
[21] “The role of bias in Neural Networks” [online] Available: https://www.pico.net/kb/the-role-of-bias-in-neural-networks
[22] “Autopilot”, [online] Available: “https://www.tesla.com/autopilotAI”
[23] Chitta Ranjan, “Understanding Dropout with the Simplified Math behind it”, [online] Available: https://towardsdatascience.com/simplified-math-behind-dropout-in-deep-learning-6d50f3f47275
[24] Madhushree Basavarajaiah, “Maxpooling vs minpooling vs average pooling”, [online] Available: https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-poolin G-95fb03f45a9

Images Used

[25] Aditya Sharma, “Convolutional Neural Networks in Python with Keras, ”[online] Available: https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python.
[26] Li Yin, “A summary of Neural Network Layers”, [online] Available:

https://medium.com/machine-learning-for-li/different-convolutional-layers-43dc146f4d0e.
[27] “Figure 2” [online] Available: https://www.researchgate.net/figure/Illustration-of-Max-Pooling-and-Average-Pooling-Figure-2-above-shows-an-example-of-max_fig2_333593451
[28] Wentong Wang, Lichun Wang, Xufei Ge, Jinghua Li, Baocai Yin, “Pedestrian Detection Based on Two-Stream UDN”, [online] available: https://www.mdpi.com/2076-3417/10/5/1866/htm
[29] “Figure 2”, [online] Available: https://www.researchgate.net/figure/Example-of-ReLU-activation-function_fig5_331800773