Using and Exploiting Eyeballer CNN

Samuel Murtaugh
Clemson University

Eric Hager
Clemson University

Lee Jackson
Clemson University

Abstract

Eyeballer is a convolutional neural network based penetration testing tool that is designed to assist a tester in narrowing the attack surface of a website. By identifying certain specific types of web pages that may be more susceptible to exploitation such as custom error pages, login pages, old looking websites, webapps and parked domains. Research has shown these types of web pages introduce vulnerabilities at a much greater rate and so the Eyeballer tool helps to identify them using a convolutional neural network (CNN). There are also vulnerabilities within CNNs, such as the weakness of backpropagation. In this study, we attempt to use the Eyeballer tool to test a large data set of screen shots of various web pages in an effort to gauge it's precision and efficiency. We also attempt to exploit some of the known weaknesses of CNNs to force the Eyeballer tool to produce false negatives or false positive results.

1 Introduction

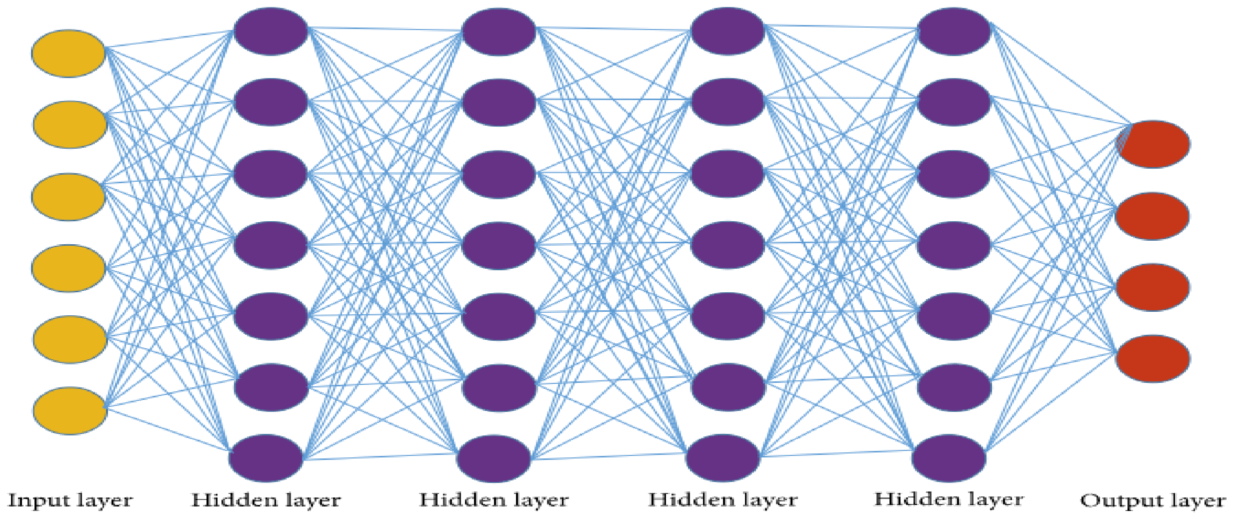
The rise of artificial neural networks (ANNs) has been a big advancement in the field of machine learning recently. These ANNs are based on the biological design of neurons in the human brain and have proven to be far more successful at performing common machine learning tasks than previous forms of artificial intelligence [5]. The basic design of an ANN consists of a large number of compute nodes which are referred to as neurons and are generally arranged in multiple layers. An input layer of nodes which takes in an input and distributes the input to hidden layers where decisions are made based on whether the input detriments or improves the final output and applies weights to the inputs accordingly. There is also an output layer where the decisions and weights applied to the inputs can be produced as a result. This basic structure shown in Figure 1 is considered to be a basis for multiple types of ANNs including feed-forward neural networks (FNNs), Restricted Boltzman Macines (RBMs), and Recurrent Neural Networks (RNNs) [5] [4].

Traditional ANNs have some associated limitations, most notably they struggle with large computationally complex sets of data which makes them particularly less efficient at image classification. CNNs, however, while they are a form of ANN, are much better suited for these tasks and are the primary neural network type used in pattern and image recognition.

Most of the existing research and CNN based tool sets designed for security in computing systems are focused on defensive security measures such as web attacks detection and intrusion prevention systems. Many variants of neural networks exist, but the most competitive ones that are used for web attack detection primarily are the Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN) [3]. Eyeballer, however, was conceived by Dan Petro and Gavin Stroy of Bishop Fox who are hacking enthusiasts. Because of their penchant for breaking into systems, they created Eyeballer as an offensive security weapon which could assist in identifying possible vulnerabilities in websites that they could target [2].

There are weaknesses in reinforcement learning systems and specifically in CNNs that have, over time, been exposed and exploited by attackers. Many CNNs are vulnerable to adversarial attacks in the form of adding tiny perturbations to inputs which can lead to incorrect results in the form of false negatives or false positives [1]. While evaluating the performance, usability and accuracy of the Eyeballer results is a large part of what this research project is designed to acheive, we would also like to attempt to exploit any possible weaknesses that Eyeballer may have built in to itself inherently given the fact that it is also a CNN.

Figure 1:



2 Related Work

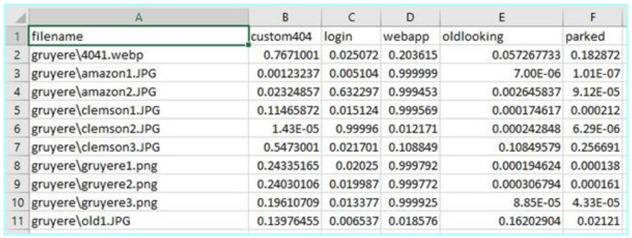
The main inspiration for this project came from Michael Kissner's "Hacking Neural Networks: A Short Introduction." Specifically, we utilized the section focused on extracting information from neural networks. Kissner posits that "neural networks have a 'memory' of what they have been trained on." A nefarious party should be able to use this memory to extract information from the network that resembles the data it was trained with; Hayes et al. and Hitaj et al. both leveraged Generative Adversarial Networks that showed this to be true. Kissner explains that one can misuse backpropagation to train a network to generate false positives that wouldn't fool a human performing the same task but will fool the neural network. He claims that the process can be illustrated as some nefarious party adding an employee to the front of an assembly line who asks his neighbor what and how much material he should pass to him, thus reverse-engineering some item that passes for the product [4].

Regarding the Eyeballer CNN itself, we were unable to find any scholarly research that utilized Eyeballer. Indeed, the network itself was not proposed in any published research paper.

3 Methodology

3.1 Installing Eyeballer

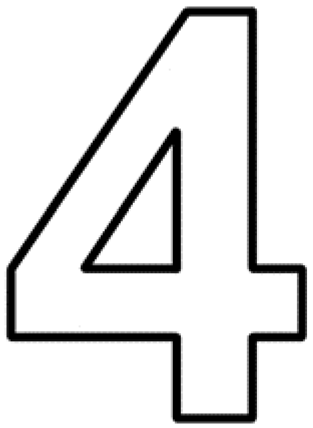
Installing Eyeballer looks quite easy if you just read its README, but it can be quite tricky. Eyeballer has a long list of requirements, one of which is tensorflow. Eyeballer's README will attempt to have you install tensor flow with a simple "sudo pip3 install -r requirements.txt", however doing this has a good chance of messing up your python installation. Using pip and sudo together is explicitly warned against in the tensorflow documentation. When installing, the directions directly from the tensorflow documentation should be used. The rest of the requirements can be installed with the pip command; I did not find the use of sudo to be necessary. In order for Eyeballer to function properly, you must also download the pretrained weights that are on the github, or train the model yourself. For the purposes of our project, we simply used the pretrained weights for testing. Eyeballer also needs its dataset downloaded and extracted in order to function, as the dataset provides labels. A link to the kaggle page for the dataset is available in the README. Upon installing eyeballer, a short set of just 11 photos of various popular websites (the clemson website, amazon, as well as an old, vulnerable web app called Google gruyere) was used to validate that Eyeballer worked. The results of this brief test are all shown in Figure 2. It seemed to struggle to detect old websites based on this limited test, and was less confident in its detection of custom 404 pages. It was very accurate and confident in its detection of webapps.

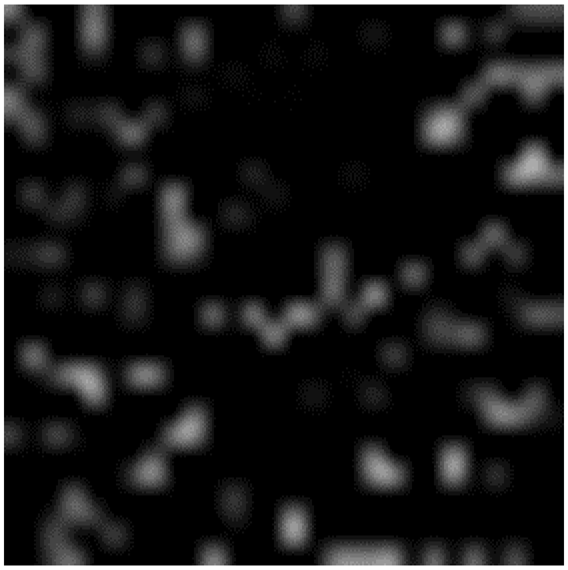


	A	B	C	D	E	F
1	filename	custom404	login	webapp	oldlooking	parked
2	gruyere\4041.webp	0.7671001	0.025072	0.203615	0.057267733	0.182872
3	gruyere\amazon1.JPG	0.00123237	0.005104	0.999999	7.00E-06	1.01E-07
4	gruyere\amazon2.JPG	0.02324857	0.632297	0.999453	0.002645837	9.12E-05
5	gruyere\clemson1.JPG	0.11465872	0.015124	0.999569	0.000174617	0.000212
6	gruyere\clemson2.JPG	1.43E-05	0.99996	0.012171	0.000242848	6.29E-06
7	gruyere\clemson3.JPG	0.5473001	0.021701	0.108849	0.10849579	0.256691
8	gruyere\gruyere1.png	0.24335165	0.02025	0.999792	0.000194624	0.000138
9	gruyere\gruyere2.png	0.24030106	0.019987	0.999772	0.000306794	0.000161
10	gruyere\gruyere3.png	0.19610709	0.013377	0.999925	8.85E-05	4.33E-05
11	gruyere\old1.JPG	0.13976455	0.006537	0.018576	0.16202904	0.02121

3.2 The Exploit

The exploit we hoped to replicate on Eyeballer was one described by Michael Kissner in Hacking Neural Networks: A Short Introduction. We tried to adapt the code he uploaded to his github for Attack 2: Extracting information to work on Eyeballer. The goal of this attack is to create a neural network that can be used to generate a photo that causes eyeballer to provide a certain output label, even if the photo does not resemble a web page that would provide that label in any way. The way the attack works is really quite simple. An example of this type of attack and output is shown below. Two pictures are present; One is obviously a four, but the other resembles static. Both of these photos will cause the implementation of MNIST built into Keras to return 4.





The way it works is quite simple. The target model is loaded into memory, and is made static and untrainable in Keras. Then, a second network is attached to the end of the target network. The layers of this network are the reverse of the target. So in the above example, it takes in the label produced by MNIST, and works its way to producing an image. The network needn't be as deep as the initial network, as it only needs to produce one image that works. As the model trains, the first half that represents the target is untrainable, and so it will not change. The second half of the model will learn how to generate a picture based on information sent to it by the first.

3.3 Our Attempt

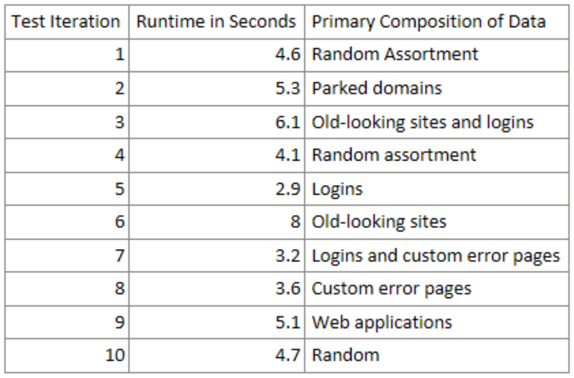
Kissner was kind enough to provide a script that performed his exploit on MNIST, and we attempted to adapt that script to work for our needs. The first thing we had to do was add code to save the eyeballer model as an h5 file to the eyballer model code. We did this by adding a single line into the model initialization function in eyeballers model.py script. After we had the h5 file, we loaded it into the attack script in place of the MNIST model. The code that followed had to be changed to fit eyeballer. Instead of 10 digit labels, the input vector had to be 5 web page labels. The initial layers of the attack network had to be changed to reflect eyeballers image dimensions. Eyeballer works on 224x224 images. Unfortunately, for some reason, eyeballer uses a network layer that is 224x224x3 instead of just 224x224 in size. We were never able to figure out why, and as a result, our script was never able to produce an image as output, although it does train the model. The expected output array of image pixels is 3 times the size it should be, so we aren't able to neatly reshape it into an image at all, much less one of the size eyeballer expects. Since we were never able to get the final stage of the script that creates the image working either, we were never able to test if the section of the script that provides the target label for the output image is working either. We simply didn't have an image to feed back into eyeballer to see how it classified them. Another mitigating factor in our ability to produce this model/script was the fact that it is very difficult to get CUDA and Keras working properly on a GPU. We initially tried only to focus on the CPU, but it would take many hours to train in that scenario. Once we did get things working on the GPU, we were plagued by memory allocation errors that forced us to severely limit the batch size attack network trained with.

4 Evaluation

Our evaluation of Eyeballer was limited primarily to testing the effectiveness of the CNN-based tool in detecting potential vulnerabilities in a collection of website screen shots that we gathered from various sources on the internet. We focused our screen shot collecting efforts on locating web pages that fit the categories that Eyeballer is specifically trying to detect so that we can get some metrics on the accuracy of the detection results as well as an evaluation of the performance of Eyeballer when running the detection. We collected 100 screenshots of old looking sites, custom error pages, parked domains, web apps and login pages as these are the categories that are detected by Eyeballer as being more likely to contain vulnerabilities that can possibly be exploited by would-be hackers and adversaries.

4.1 Runtime Performance

Eyeballer's performance was judged based on an average runtime during an evaluation of 100 random screen shots. We chose a different set of images for each performance evaluation and evaluated the total runtime of the Eyeballer program while running detection on 10 different sets of images. We found Eyeballer overall to be quite efficient at running detection and image classification with an average runtime of just 4.76 seconds. The longest runtime experienced was 8 seconds and the shortest runtime was a blazing fast 2.9 seconds. The runtime test results can be observed in Figure 3.



Test Iteration	Runtime in Seconds	Primary Composition of Data
1	4.6	Random Assortment
2	5.3	Parked domains
3	6.1	Old-looking sites and logins
4	4.1	Random assortment
5	2.9	Logins
6	8	Old-looking sites
7	3.2	Logins and custom error pages
8	3.6	Custom error pages
9	5.1	Web applications
10	4.7	Random

Figure 2

4.2 Accuracy of Results

The accuracy of Eyeballer's detection and image categorization abilities was another area we wanted to evaluate. Since the results that are output by the program are in the form of a confidence score out of a total of 1.0 possible for full confidence, we had to determine a threshold for a reasonable level of confidence in the label that was applied to a given image. Though this may be slightly subjective, we felt that the confidence score should be above 0.9 in order to indicate that Eyeballer was reasonably certain that the label being applied to the image was an accurate classification. Eyeballer's documentation does not provide any specific metrics in this regard.

4.2.1 Custom Error Pages

The Eyeballer command line detection and image classification of custom error pages was fairly successful and particularly impressive given the complexity of some of the custom error pages we chose to analyze. Many of these sites were found on various Internet lists of the most interesting custom error pages and as a result they are highly customized and often visually interesting. This did not seem to affect Eyeballer's ability to classify most of them with a reasonable degree of confidence. Out of a total of 36 custom error pages chosen as a subset for analysis, we found Eyeballer was able to identify 16 of them with a greater than 0.9 confidence score. This was by far the best rate of detection among the categories that Eyeballer is listed as being capable of detecting. 26 out of the 36 were identified with a greater than 0.75 confidence score and all but 7 of the images were identified with a greater than 0.5 confidence level. The Eyeballer tool proved to be formidable at identifying and labeling custom error pages based on all of our analysis.

4.2.2 Parked Domains

The command line detection and labeling of parked domains was slightly less accurate than the custom error page image classification. We ran the detection on a subset of images of 50 sites that were parked domains and found that many of the confidence scores did not reach the 0.9 threshold for high accuracy. Out of a total of 50 sites that we analyzed, only 11 of the images were identified with a greater than 0.9 confidence score. A total of 20 sites were identified with a greater than 0.75 confidence score and 12 of the sites were considered to be missed classification as they fell well below the 0.5 confidence score that would be a reasonable level of any detection.

4.2.3 Old Looking Sites

When running the command line detection and image classification on a subset of the data that contained a group of 40 old looking websites, curated from various internet lists of old websites that still exist in their original format, we found Eyeballer to be quite inefficient at detecting these types of images. Given the 0.9

confidence score threshold we determined to represent a high degree of confidence in the label classification, only 1 out of the total of 40 old looking websites was classified with this level of confidence. Only 2 of the 40 images were labeled with greater than a 0.64 degree of confidence, and only 5 of the sites were labeled correctly with greater than a 0.25 confidence score. In our evaluations, Eyeballer seems to struggle mightily at categorizing old looking sites appropriately. This may lead to many missed potential vulnerabilities in the field when being utilized by professional penetration testers. With our previously determined metric of a 0.9 confidence score being considered a highly accurate detection confidence, we found that Eyeballer was barely above the 2% mark in accuracy of detection of old looking web sites. A subset of the top results sorted by confidence score can be seen in Figure 4. It is also worth noting the high confidence scores some of these images were given in regards to them being custom error pages, which they were not. As an additional test, and to make sure we were not getting inaccurate results due to the particular data set we chose to analyze, we selected another set of web pages from the Wayback Machine Internet Archive of the top 20 most-visited web sites from 1999. The results were equally unfulfilling and wildly inaccurate. Only 3 of the sites were detected and labeled with a greater than 0.1 confidence score. Eyeballer itself describes an old-looking site as having "blocky frames, broken CSS, that certain 'je ne sais quoi' of a website that looks like it was designed in the early 2000s. You know it when you see it" [2]. Since we know that all of these websites were designed even before the early 2000s, they should definitely be designated as old-looking. One reason that this could happen is that these popular websites were all created with forward-thinking design concepts. However, all of the examples looked very dated in the opinions of our team members.

4.3 Web-based Eyeballer 2.0 Evaluation

Towards the end of our research, as this project was coming to a close, Bishop Fox released a web-based version of their Eyeballer tool on November 15, 2021, dubbed Eyeballer 2.0. We figured it would be prudent to do a brief evaluation of this new version to give a fully comprehensive picture of the Eyeballer landscape. This version of the tool is just a simple web interface with a file upload section where a tester can drag and drop any number of images of their choosing. It is noted on the site that no files are actually uploaded anywhere and no data leaves the user's computer at all during the process. This may or may not contribute to the overall reduction in runtime performance when compared to the command line version of Eyeballer we had evaluated in the previous sections. We first selected a set of 36 images that we had previously analyzed in the command line tool. This was the same group of custom error pages in our previous analysis. Since this group of images gave us our best results in the command line tool, we felt it was a good measuring stick with which to guage the precision and performance of Eyeballer 2.0. For reference, when we ran setection in the command line, it completed in under 2.5 seconds on this data set. The web-based 2.0 version of Eyeballer has greatly reduced performance and increased the time required for labeling images. The set of 35 images took over 17 seconds to classify using the web-based version of Eyeballer. That is an increase in runtime of over 600%. As far as precision is concerned, the web-based version of Eyeballer was only able to accurately label 7 out of the 36 images as custom error pages. Considering the fact that the command line tool was able to detect 16 of the same images with a greater than 0.9 confidence score, we can only determine that the web-based version of Eyeballer is not only slower at processing images, but also less accurate at classifying them correctly. Ultimately, this is a relatively new iteration of the Eyeballer tool and we expect Bishop Fox will work on making it faster and more accurate in the future.

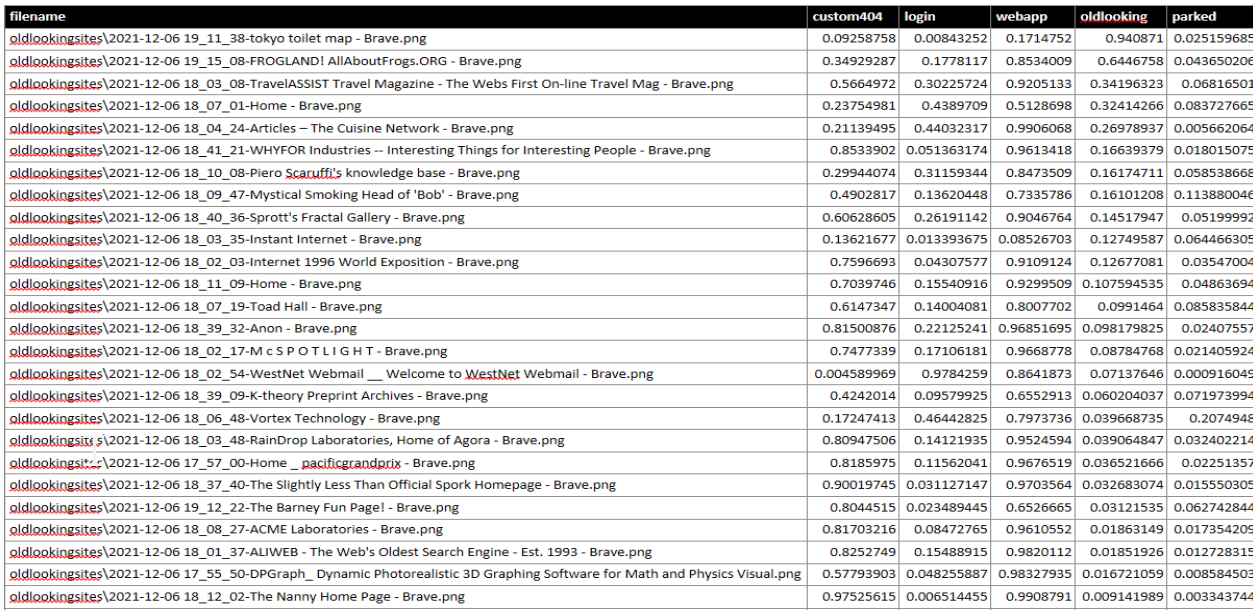
5 Future Work

The first item of future work would be to continue analyzing Eyeballer's network structure and tweaking our attack network structure to determine how to get our exploit working. As of right now, our error seems to be in the final stage of the network. Eyeballer takes in 224x224 images, but for some reason, works on data that is 224x224x3. Our attack network correspondingly outputs data that is 224x224x3, and we cannot translate it into a picture that is 224x224.

Other areas for future work include attempting the rest of the attacks described in Kissner's work[3]. Kissner describes 8 attacks, and we only examined one of them. Attempting all of them on Eyeballer could prove an educational exercise. It would be interesting to see if Eyeballer was more susceptible to the rest of them than it was to the one we chose.

The particular exploit requires access to the model to work. It would be interesting to attempt to create a GAN that could learn to "trick" Eyeballer without the need for access to the model. The only reason that wasn't attempted here was because the grouped lacked the Neural Networking expertise to create a GAN.

Since this exploit was predicated on having access to



filename	custom404	login	webapp	oldlooking	parked
oldlookingsites\2021-12-06 19_11_38-tokyo toilet map - Brave.png	0.09258758	0.00843252	0.1714752	0.940871	0.025159685
oldlookingsites\2021-12-06 19_15_08-FROGLAND! AllAboutFrogs.ORG - Brave.png	0.34929287	0.1778117	0.8534009	0.6446758	0.043650206
oldlookingsites\2021-12-06 18_03_08-TravelASSIST Travel Magazine - The Webs First On-line Travel Mag - Brave.png	0.5664972	0.30225724	0.9205133	0.34196323	0.06816501
oldlookingsites\2021-12-06 18_07_01-Home - Brave.png	0.23754981	0.4389709	0.5128698	0.32414266	0.083727665
oldlookingsites\2021-12-06 18_04_24-Articles – The Cuisine Network - Brave.png	0.21139495	0.44032317	0.9906068	0.26978937	0.005662064
oldlookingsites\2021-12-06 18_41_21-WHYFOR Industries -- Interesting Things for Interesting People - Brave.png	0.8533902	0.051363174	0.9613418	0.16639379	0.018015075
oldlookingsites\2021-12-06 18_10_08-Piero Scaruffi's knowledge base - Brave.png	0.29944074	0.31159344	0.8473509	0.16174711	0.058538668
oldlookingsites\2021-12-06 18_09_47-Mystical Smoking Head of 'Bob' - Brave.png	0.4902817	0.13620448	0.7335786	0.16101208	0.113880046
oldlookingsites\2021-12-06 18_40_36-Sprott's Fractal Gallery - Brave.png	0.60628605	0.26191142	0.9046764	0.14517947	0.05199992
oldlookingsites\2021-12-06 18_03_35-Instant Internet - Brave.png	0.13621677	0.013393675	0.08526703	0.12749587	0.064466305
oldlookingsites\2021-12-06 18_02_03-Internet 1996 World Exposition - Brave.png	0.7596693	0.04307577	0.9109124	0.12677081	0.03547004
oldlookingsites\2021-12-06 18_11_09-Home - Brave.png	0.7039746	0.15540916	0.9299509	0.107594535	0.04863694
oldlookingsites\2021-12-06 18_07_19-Toad Hall - Brave.png	0.6147347	0.14004081	0.8007702	0.0991464	0.085835844
oldlookingsites\2021-12-06 18_39_32-Anon - Brave.png	0.81500876	0.22125241	0.96851695	0.098179825	0.02407557
oldlookingsites\2021-12-06 18_02_17-M c S P O T L I G H T - Brave.png	0.7477339	0.17106181	0.9668778	0.08784768	0.021405924
oldlookingsites\2021-12-06 18_02_54-WestNet Webmail __ Welcome to WestNet Webmail - Brave.png	0.004589969	0.9784259	0.8641873	0.07137646	0.000916049
oldlookingsites\2021-12-06 18_39_09-K-theory Preprint Archives - Brave.png	0.4242014	0.09579925	0.6552913	0.060204037	0.071973994
oldlookingsites\2021-12-06 18_06_48-Vortex Technology - Brave.png	0.17247413	0.46442825	0.7973736	0.039668735	0.2074948
oldlookingsites\2021-12-06 18_03_48-RainDrop Laboratories, Home of Agora - Brave.png	0.80947506	0.14121935	0.9524594	0.039064847	0.032402214
oldlookingsites\2021-12-06 17_57_00-Home _ pacificgrandprix - Brave.png	0.8185975	0.11562041	0.9676519	0.036521666	0.02251357
oldlookingsites\2021-12-06 18_37_40-The Slightly Less Than Official Spork Homepage - Brave.png	0.90019745	0.031127147	0.9703564	0.032683074	0.015550305
oldlookingsites\2021-12-06 19_12_22-The Barney Fun Page! - Brave.png	0.8044515	0.023489445	0.6526665	0.03121535	0.062742844
oldlookingsites\2021-12-06 18_08_27-ACME Laboratories - Brave.png	0.81703216	0.08472765	0.9610552	0.01863149	0.017354209
oldlookingsites\2021-12-06 18_01_37-ALIWEB - The Web's Oldest Search Engine - Est. 1993 - Brave.png	0.8252749	0.15488915	0.9820112	0.01851926	0.012728315
oldlookingsites\2021-12-06 17_55_50-DPGraph_ Dynamic Photorealistic 3D Graphing Software for Math and Physics Visual.png	0.57793903	0.048255887	0.98327935	0.016721059	0.008584503
oldlookingsites\2021-12-06 18_12_02-The Nanny Home Page - Brave.png	0.97525615	0.006514455	0.9908791	0.009141989	0.003343744

Figure 3

the model, learning how to ”poison” a model you have access to could be another worthwhile area to invest research time into. In particular, seeing if this method of ”attaching” a neural network to the front of another and training it could be used to force the target network to learn something. Of course, this would require changing the attack to allow the first half of the network to be trainable.

References

[1] CHEN, T., LIU, J., XIANG, Y., NIU, W., TONG, E., AND HAN, Z. Adversarial attack and defense in reinforcement learning-from ai security view. *Cybersecurity 2* (12 2019).

[2] FOX, B. Eyeballer. https://github.com/BishopFox/eyeballer, 2021.

[3] JEMAL, I., HADDAR, M. A., CHEIKHROUHOU, O., AND MAHFOUDHI, A. Performance evaluation of convolutional neural network for web security. *Computer Communications 175* (2021), 58–67.

[4] KISSNER, M. Hacking neural networks: A short introduction. *arXiv preprint arXiv:1911.07658* (2019).

[5] O'SHEA, K., AND NASH, R. An introduction to convolutional neural networks. *CoRR abs/1511.08458* (2015).

Notes

Sam Murtaugh was responsible for getting Eyeballer working initially, light initial testing of Eyeballer, attempting the exploit and writing/modifying the script for the exploit. Wrote methodology and future work.

Eric Hager was responsible for initial literature review, collecting data sets for testing and evaluation of Eyeballer, evaluating Eyeballer results and proficiency, evaluation of Eyeballer 2.0 web-based tool, and writing the Abstract, Introduction and Evaluation sections of the paper.

Lee Jackson was responsible for initial literature review, collecting data sets for testing and evaluation of Eyeballer, evaluating Eyeballer results and proficiency, writing the Related Work section, and assisting with writing the Evaluation section of the paper.