Adversarial Image Generation for Deep Learning Systems

Anonymous Bot
Department of Computer Science, University of California, Davis
1 Shields Ave, Davis, CA 95616

***Abstract*—Deep Neural Networks have become part of every day life and it is imperative that the robustness of these systems be ascertained using mathematical models. Classifiers can be made more robust by training with varying distributions. A way to do this is to create adversarial samples so as to regularize the neural network during training. In this work we combine some of the state of the art white box attacks to generate adversarial samples for Neural Networks as well as propose our own black box attack oblivious to the original Neural Network parameters. The results show that the white box attacks work for any network architecture while the black box attacks work infrequently as the network gets deeper.**

I. Introduction

Deep Neural Networks (DNNs) are considered the cutting edge technology to make decisions regarding image classification, voice recognition, advertising, bioinformatics, credit assessment, drug discovery and toxicology, etc. Accordingly, it is important to understand the threat model associated with Deep Neural Networks. Recent research [1], [2] has shown that it is possible to generate adversarial examples in speech that can be interpreted by machines but not humans. Consequently, it is possible to control someone's phone without the person's knowledge. [8] This has also been expanded to images [3], where small perturbations in the image pixels was sufficient to fool a trained Neural Network. In this regard our contribution is to formulate, using methods of Computer Vision, a way to create adversarial images for trained Deep Neural Network. The motivation for this work comes from the fact that machines can classify captchas with almost human accuracy. Our goal is to create images that can be correctly classified by humans but not by a Deep Neural Networks.

We begin by distorting the image in specific ways so as to increase the error rate of the targeted deep neural network. Initially we test our samples on a three layer Deep Neural Network to gain more intuition into the learning process and use the templates the Network learns at the end as a way to create additional adversarial samples. In our first attack approach, we train another neural network, called an Adversarial Transformative Network (ATN) [13], to generate distortions in the input image. Existing ATN implementations don't restrict the distortions to be within a user-defined neighborhood. Our work, on the other hand, modifies the behavior of an ATN to restrict it to a neighborhood. For our second attack approach, we extend our idea to break a Deep Convolutional Neural Network (CNN), where we don't use the parameters of the target Deep CNN.This attack is oblivious to the target CNN parameters like architecture and weights.

II. Background

In this section we summarize existing techniques for generating adversarial images and also explore their limitations.

2.1 Adversarial Examples

Szegedy et al. first introduced the term "adversarial examples" in 2014; that is, inputs with slight perturbations which trick modern neural network models into misclassifying data. [4] This indicates that even cutting edge neural networks which perform excellently on the test set, utterly fail compared to humans, when the data is perturbed slightly. These adversarial examples are particularly robust, which means adversarial examples generated on one neural network can trick another neural network trained

with completely different parameters. [5] In certain datasets, including the ImageNet [6] dataset, the adversarial examples were almost indistinguishable to the human eye. This clearly points out a security threat associated with machine learning models.

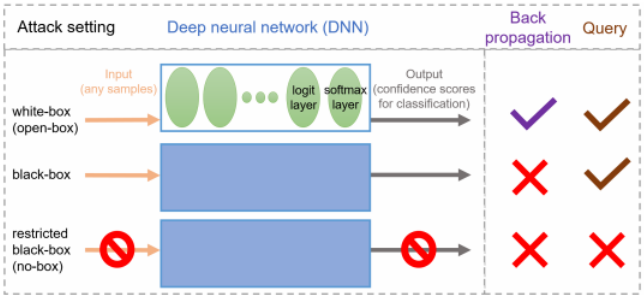


Fig. 1. Overview of three different types of attacks on DNNs.In the black box attack, the attacker can input an example and only observe the output(Query), while in the white box attack the attacker can also access the internal parameters of the DNN(Back Propagation). [10]

2.2 Types of Attacks:

Current adversarial attack methods can be divided into several categories based on what information they have access to. In a white box attack, the attacker has access to the weights and the architecture of the target network. This can also include attacks which have access to the data which the target network was trained on. In a black box attack, the attacker only has access to the final output of the network. For an image classifier, this would be the predicted class of the image. Furthermore, some attacks seek to simply cause the image to be misclassified, while others seek to have the perturbed image classified as a specific class. An overview of these attacks is given in Figure 1. [14]

2.3 Linear explanation of Adversarial Examples and the Fast Gradient Sign Method

In 2015, Goodfellow et al. at Google Inc. hypothesized that the underlying cause of neural networks not being able to correctly classify adversarial examples is due to their inherent linearity. This essentially means that adversarial examples can be generated for simple linear models if its inputs have sufficient dimensionality. In many problems the input precision is limted. For instance, for an image using 8-bits per pixel, the model would discard all information below 1/255 of the dynamic range. Thus an input image **x** would be considered to be same as an adversarial input **x' = x +** η by the classifier. An example is shown in Figure 2. They also demonstrated a method of generating adversarial examples, the "fast gradient sign method". In this method adversarial perturbations can be quickly and easily generated by adding a small vector along the direction of increasing the cost. [5]

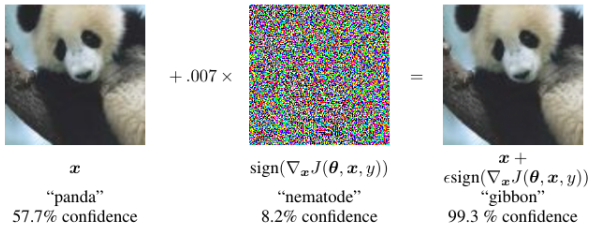


Fig. 2. In 2015, Goodfellow et al. demonstrated adversarial example generation applied to GoogLeNet on ImageNet. This demonstrates the Fast Gradient Sign Method. Here .007 is the magnitude of the smallest bit of an 8-bit image encoding. [5]

Although the FGSM method is fast and efficient, it has been shown in later works to be easily defended against. Despite this, FGSM gives the clearest intuition on generating adversarial examples, and has inspired most other techniques. Even our work derives from this method's idea of confining perturbations within a neighborhood.

2.4 JSMA

Jacobian-based Saliency Map Attack (JSMA) is an attack optimized under L_0 introduced by Papernot et al. Similar to FGSM, this method seeks to make small changes in the input image to produce a perturbed image. This method seeks to modify only certain features of the input and create a perturbed input which will be classified as a target class. The main innovation in this approach is the ability to find the important features in the input, which when modified, will lead to a target classification by the model. To do this, we can construct an adversarial saliency map. [7] [8]

JSMA uses an adversarial saliency map, shown in Figure 3, which shows the best features to be perturbed in an input to generate the desired classification. This saliency map is constructed using the forward derivative of the target network. High values in the saliency map correspond to features that will increase the probability of the input being classified as a target class or decrease the probability of the input being classified as another classes.

After obtaining the saliency map, we can use it to perturb the given input. There are two new parameters to keep in mind here. First, θ defines the amount by which a feature is perturbed. Second, υ defines the maximum number of epochs that the algorithm can run for. Both of these parameters serve to minimize the distortion to the image and should be adjusted depending on the specific problem.

JSMA searches for a pair of pixels which can be modified to increase the likelihood of the target class and reduce that of the other classes. This has been shown to be a drawback of this algorithm. When it comes to large images, such as $1000X1000X3$, it would be computationally expensive to search through all of the pairs of pixels. [8]

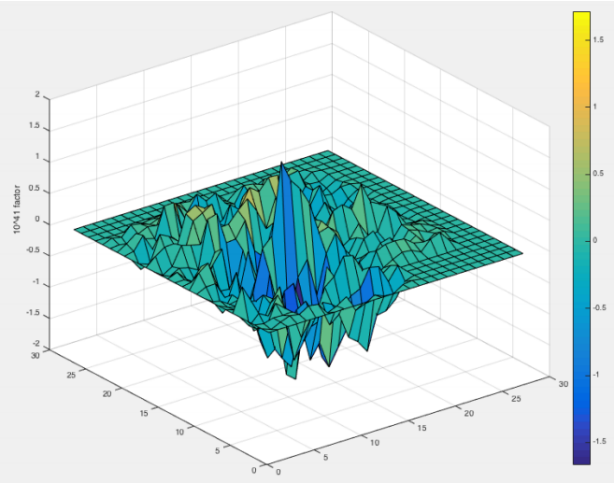


Fig. 3. Saliency map of a 28 x 28 image generated by JSMA. Those points with large absolute values have greatest effect on the classification of the image. [9]

2.5 Carlini & Wagner (C&W) Attack

In 2016, Carlini and Wagner proposed a strong white box attack. Being a white box attack they take advantage of the internal parameters of a targeted DNN and designed adversarial attacks as an optimization problem. [8] They use the Euclidean distance, which is the L_2 norm, to assess the difference between adversarial and original examples.The new image is optimized by the Euclidian norm to be close to the original image. Given an image, the C&W method generates a perturbed image which is classified as a target class. The perturbation is also optimized to have the highest probability of being classified as the target class. They also showed that their attack bypasses the ten different methods designed for detecting adversarial examples. [11] However, the defence models proposed by Aleksander Madry et al. showed favorable results against the C&W attack, as well as FGSM. [10] [12]

III. White Box Adversarial Transformative Networks:

3.1 Adversarial Transformative Networks

Adversarial Transformative Networks (ATNs) were proposed in 2017 by Baluja and Fischer in 2017 in their paper, Adversarial Transformative Networks: Learning to Generate Adversarial Examples. In contrast to other methods such as FGSM or JSMA that generate adversarial examples directly by computing gradients on the image pixels or solving an optimization problem on them, ATN is by itself a neural network that transforms an input into an adversarial example against a target network or a set of networks. ATNs can be targeted or untargeted and can also be trained in a black-box or a white-box manner. [13]

We define the target neural network as y = f_w(x), with w being the parameter matrix of the network f, and x is the input. The output of f, y is a vector representing the class probabilities for each label. The ATN is formally defined as $x' = g_{f,\theta}(x)$.θ is the parameter matrix of the ATN and x' is the adversarial transformation of x, such that $x' \sim x$, but argmax f_w(x) $\neq$ f_w(x'). [13]

3.2 Cost Function

The parameters θ are obtained by solving an optimization on the cost function defined as follows:

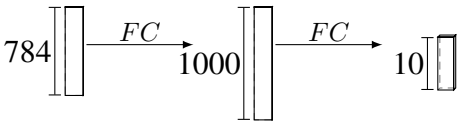


Fig. 4. Architecture of target ANN

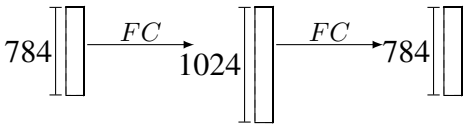


Fig. 5. Architecture of ATN

$$\sum_{x_i \in X} \beta L_x(g_{f,\theta}(x_i), x_i) + L_y(f(g_{f,\theta}(x_i)), f(x_i))$$

L_x is a measure of the difference between the generated adversarial example and the original input. $L_y$is a measure of the difference between the output generated by f on an adversarial example and a particular target output for a targeted attack. For an untargeted attack, L_y can be taken as a measure of likelihood between the output generated by f on an adversarial example to the output generated by f on the corresponding original input. In both cases, the goal is to identify values for θ that minimize both L_x and L_y, and thus, the entire cost function. β is tuning parameter to balance out the two terms.

For untargeted attacks, L_x and L_y can both be represented by straightforward L_1 or L_2 costs. For targeted attacks, L_x can be a simple straightforward L_1 or L_2 loss, but L_y would have to be modified to bias the learning towards the targeted task. To do this, the definition of L_y is modified to include a re-ranking function. Thus we could have a definition such as,

$$L_y = L_y(f_w(g_\theta(x_i)), r(f_w(x), t)$$

Here, r is the re-ranking function, and t is the target class that we're trying to make the classifier output. The choice of $r()$ can be used to tune the rate of training. For instance, r can return a vector containing just a one-hot encoding, such that only the value corresponding to the target class is 1, with all others being 0. This approach however, does not make use of the probabilities that are computed by f to drive the biasing. Hence, a better re-ranking function would be,

$$r(f_w(x), t) = norm(\alpha * max(f_w(x))); k = t,$$

$$r(f_w(x), t) = norm(f_w(x)); otherwise$$

Where, $norm$ is a standard normalization function, k represents the possible output classes and $max()$ returns the max value of its input vector. $\alpha > 1$, is an additional parameter that ensures that the reranked output points only towards one particular class. [13] For example, if an output of a target neural network outputting three labels was (0.1, 0.2, 0.3), then after re-ranking, it would become (0.6, 0.2, 0.3), before normalizing. Here, the target label is the first label and the α parameter's value is 2. This vector gives more weight to the target label than a one-hot vector, which would just be (1, 0, 0)

3.3 Our Approach

Once the parameters are obtained, the ATN can be used to generate a perturbation to each node of the input, thereby giving a new adversarial input. One point to note in the existing work on adversarial transformative networks is that, it does not restrict the input perturbation to lie within some defined neighbourhood, ε, as used in FGSM. While the inputs are made to look as similar as possible by using the L_x term in the cost function, there is no explicit bound on these perturbations, in terms of a neighbourhood. Thus in our experiments, we restrict the output nodes of the ATN to lie in the range [-1, 1]. These output nodes, multiplied by ε and added to the inputs would restrict the perturbations to be within a neighbourhood ε. This is mathematically formulated as:

$$x' = x + \varepsilon * g_{f,\theta}(x)$$

Since $g_{f,\theta}(x)$ is quantifying the extent of perturbation as opposed to the actual value of the input after the perturbation, we modify the definition of the cost to have L_x only include $g_{f,\theta}(x)$, and L_y to include$f(x+g_{f,\theta}(x))$ as opposed to just $f(g_{f,\theta}(x))$. Thus, the cost function, taking all choices into account would be:

$$\sum_{x_i \in X} \beta L_x(g_{f,\theta}(x_i)) + L_y(x_i + f(g_{f,\theta}(x_i)), f(x_i))$$

3.4 Training

Training the ATN only requires the data that is to be perturbed, and knowledge of the targeted network. Knowledge of the targeted network requires both the architecture of the trained network as well as all the weights used by this model. The original labels used to train the targeted network are not required to do this training.

Training involves standard forward propagation and backward propagation on the ATN, by computing the derivatives of the cost function defined above with respect to the parameters θ of the ATN. Although derivatives with respect to the parameters w of the target network are not required, these parameters are used as part of the chain rule while computing the derivatives with respect to θ. Computing the derivatives of L_x is similar to what is followed in the standard back propagation to compute derivative of g with respect to θ. Derivative of L_y requires differentiating f with respect to θ. This is done by differentiating f with respect to g and g with respect to θ. Computing the derivative of f with respect to g is the same as what is done in FGSM, since g is the input to f in L_y.
Mathematically,

$$\frac{\delta L_y}{\delta \theta} = 2 * (f_w(x + g_{f,\theta}(x)) - r(f_w(x), t) * \frac{\delta f}{\delta \theta}$$

$$\frac{\delta f}{\delta \theta} = \frac{\delta f}{\delta g} * \frac{\delta g}{\delta \theta}$$

Parameters θ in each layer of the ATN are updated in every iteration like with standard gradient descent.

3.5 Experiments

We generated perturbed inputs from the non-MNIST dataset. Each image in the non-MNIST dataset is a 28x28 pixel image, giving a total of 785 features in the input, including the bias. We first ran our experiments to break a simple 3 layer feed forward neural network. As mentioned above and seen in Figure 4, the input layer consists of 785 nodes. These are mapped onto the hidden layer that has 1000 nodes. The 1000 nodes of the hidden layer are then mapped to 10 nodes in the output layer, with each node representing the probability of the input corresponding to a particular class (classes are alphabets from 'A' to 'J'). Sigmoid activations were used in both the hidden, as well as the output layers. Upon training with 20,000 samples, this feed forward neural network had an error of 0.34 on the training set.

The ATN used to break the network described above was also a 3-layered feed forward neural network. As seen in Fig. 5, it has 785 nodes in the input layer, 1024 nodes in the hidden layer and 784 nodes in the output layer. The 784 nodes in the output layer each correspond to a perturbation to each of the 784 nodes in the input layer, without the bias term. We did not have any activations in the hidden layer and used $tanh$ activation in the output layer. As mentioned earlier, $tanh$ activations are used in the output layer to restrict the perturbations to be within the range of a neighbourhood, ε. We trained this model with a few different combinations for the choice of the reranking function between what was described above and a one hot encoding, and also the choice of using L_1 and L_2 for both L_x and L_y. We were easily able to break this simple feed forward neural network, and increased the error rate to 0.9, regardless of the choice of target in each of these different combinations.

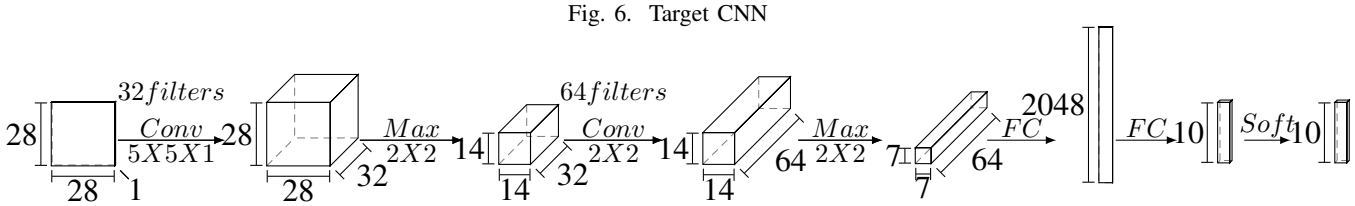
IV. Black Box Adversarial Transformative Networks

In this section we extend our work to a targeted black box attack setting with Deep CNNs.

4.1 Target CNN

As before, we define a target network as $y = f_w(x)$ where w is the parameter matrix of the network f, and x is the input. Here f is a Convolution Neural Network, having 2 convolution layers, 2 pooling layers and 2 fully connected layers. The input image is a 28 x 28 matrix which is convolved with 32 $5x5$ filters with same padding such that the output is same as the input(in size per activation map). Afterwards, we apply max-pooling with a $2x2$ kernel, with a stride of 2. This is followed by a second

Fig. 6. Target CNN



convolution layer with 64, $5x5$ filters using same padding as before, followed by another max-pooling operation with a $2x2$ kernel, with a stride of 2. It should be noted that the outputs after the convolution operation are passed through an activation function before the max-pooling operation. After this, the activations are flattened and passed to the first fully connected layer with 2048 hidden units, which is followed by passing the activations down to another fully connected layer with 10 nodes corresponding to the 10 character classes (A through J). This is then passed to softmax with 10 units to calculate the probabilities associated with each class.

4.2 Deep Convolutional ATN

The ATN as defined before

$$x' = g_{f,\theta}(x)$$

is modified for the black box attack. Previously, we had access to the target Neural Network(f) parameters and we could compute the derivative of f with respect to g(ATN), but in black box attack these parameters are unknown and we need to modify the ATN definition to exclude this parameter. As such our new definition of ATN is

$$x' = g_{\theta}(x)$$

. This represents an independent CNN where the number of input layer and output layer nodes are same, and we use it to find an encoding of the input image that is adversarial on the target image. The architecture of the Convolutional ATN is as follows: In total we have 4 convolutional layers. As before the size of input images is 28 x 28 which is convolved with 32, 3 x 3 filters. Note that we don't pass the output of the convolution layer through an activation function and we don't use subsampling either(since the output is an image and not a set of probabilities). This is followed by another convolution layer in which the outputs from previous convolution layer are convolved with 64, 3 x 3 filters. Usually after the traditional convolution layers, a traditional CNN will have a couple of fully connected layers. In this experiment, in order to mimic this architecture, we use 1 x 1 convolutions. This allows us to train the network faster and gives better results. We first convolve the outputs of the second convolution layer with 64, 1 x 1 filters, this is followed by convolution with 1, 1 x 1 filter. Note that the last convolution only uses 1 filter and after the convoltion we pass it through an activation function i.e tanh() function.

4.3 Cost Function

The cost function is modified from before to account for our black box setting as follows:

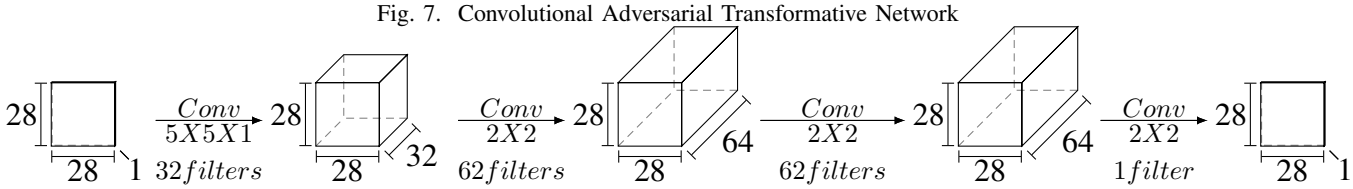
$$\sum_{x_i \in X} \beta L_x(g_\theta(x_i), x_i) + L_y(f(g_\theta(x_i)), f(x))$$

where L_x as before is the difference between the generated adversarial example and the original input. Additionally, since this is a targeted attack, L_y has to be tuned to nudge the Convolutional ATN towards learning a targeted adversarial example. As such L_y is modified from the earlier version as follows:

$$L_y = L_y(f(g_\theta(x_i)), r(f(x), t)$$

where r is the re-ranking function as described before. Intuitively L_x represents the amount of distortion in the adversarial sample compared to the original sample and L_y represents the difference between the output probabilities of the final layer for our adversarial example when passed through the target network and the output probabilities we want it to have biased towards the target class(say we want A classified as C).

Fig. 7. Convolutional Adversarial Transformative Network



4.4 Training

To train the C-ATN, we first find the encoding of an input sample using the C-ATN which is the function $g()$, then we input this encoding into the target CNN which is the function $f()$. This gives the likelihood of how the target network classifies our encoding of the input sample. In this attack, the C-ATN only needs to be able to query the target network, no additional information of the target network is needed.

We find the derivatives of the cost function with respect to θ and proceed with standard back propagation. Training can be done in batches or by samples individually. Computing the derivatives of L_x is similar to what we did in the last section. For the L_y part, we need to query the target CNN to get the value $f(g_\theta(x_i))$. Using these values, the re-ranking function determines based on a target class what values the target network should output for our adversarial sample. The difference between the two probability vectors is the loss. As an example, if a sample belongs to class 1, the target network will output probabilities close to the following vector: $f(x') = [1, 0, 0]$ (for a 3 class classification network). If we want the sample to be classified as a class 2 sample, the re-rank function will output something like this $r(f(x'), t) = [0, 1, 0]$. The difference of the two vectors is the loss which we need to minimize. Notice that this is one-hot encoding(just for demonstration), in practice we use actual probabilities.

4.5 Experiments

For the CNN, we used standard batch gradient descent with batch size 150. In total our training set had 20,000 samples and in 2000 iterations we were able to get the training accuracy to about 99.4 percent and our cross validation accuracy was about 93.5 percent.

For the C-ATN part, we trained the network with a single sample each time. Everytime the sample moves through the network, the network creates an adversarial encoding of the sample, which is then used to query the original CNN which gives us what the target CNN thinks of this sample. Afterwards, we combine the loss as described before for both the distortion from the original sample, as well as the likelihood of the current sample being classified as the target adversarial sample and then back propagate on the C-ATN. Generally it took about 2000 iterations to generate each target adversarial sample.

V. Results

With the white box attack, we executed different combinations for L_x and L_y among L_1 and L_2. We also executed different re-ranking functions and compared our neighbourhood approach against the classical approach. Despite running all this against different targets we almost always obtained a high error rate of 0.9. In the black box setting, the targeted attack seems to work(for the most part) for another class than the target. For instance, a sample that was originally B and we set the target to be A, after finding the adversarial encoding of the sample, it was classified as H by the target CNN. The attack seems to work for a different target than the one intended. The attack works for the intended target sparingly.

VI. Discussion

The results of our experiments show that for simple ANNs, regardless of the choice of the cost function, re-ranking function or the activations used in the ATN, breaking the simple target ANN was not very difficult. This can be attributed to the lesser complexity of simple ANNs. This is not true for Deep Convolution Networks especially in the black box setting where the attacks work only

Fig. 8. The left most image represents the original image, we set the C-ATN target as class A(i.e generate adversarial samples that are classified as A). The next image is the perturbed image that was classified as D by the target CNN, the image to its right was classified as A, the next image was classified as I, the image to its right was classified as H, the next image was classified as C, the following image to the right was classified as G and D with almost equal probabilities, and finally the last image above had probabilities equally distributed for more than 5 classes.

Fig. 9. The left most image represents the original image, we set the C-ATN target as class A(i.e generate adversarial samples that are classified as A). The next image is the perturbed image that was classified as I by the target CNN, the image to its right was classified as A, the next image was classified as J, the image to its right was classified as G, the next image was classified as D, the following image to the right was classified as H and I with almost equal probabilities, and finally the last image above was classified as D and I with almost equal probabilities.

infrequently. The C-ATN often generates advesarial samples for other targets than the one that is intended. Also, the choice of parameters matters in this case. For the cost function, L_x is scaled down considerably and L_y needs to be multiplied by a large number for the attack to succeed, this can be attributed to the lack of knowledge of the parameters of the target networks and inability to back propagate on the original network.

VII. Conclusion

Many current methods of generating adversarial examples can be defended against. In our paper, we proposed two alternative attacks. The white box attack could access the internal parameters of the target network and yielded great results. The black box attack on the other hand did not perform as well, owing to the greater complexity of CNNs and not having access to the internal parameters of the target network. In conclusion, future work should focus on finding better cost functions for black box attacks.

VIII. Source Code

The source code of the project can be found at the following url
https://github.com/Zainul-Abi-Din/DeepLearning

Fig. 10. The left most image represents the original image, we set the C-ATN target as class B(i.e generate adversarial samples that are classified as B). The next image is the perturbed image that was classified as H by the target CNN, the image to its right was classified as B, the next image was classified as D, the image to its right was classified as J, the image to its right was classified as A and I with almost equal probabilities, the next image was classified as D and I with almost equal probabilities, the following image to the right was classified as A, D and H with almost equal probabilities.

References

[1] S. M. Moosavi-Dezfooli, A. Fawzi, P. Frossard DeepFool: a simple and accurate method to fool deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[2] S. Alfeld, X. Zhu, and P. Barford Data poisoning attacks against autoregressive models. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[3] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel Adversarial perturbations against deep neural networks for malware classification In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[4] Szegedy, Christian, Zaremba, Wojciech, Sutskever, Ilya, Bruna, Joan, Erhan, Dumitru, Goodfellow, Ian J., and Fergus, Rob. Intriguing properties of neural networks. In *ICLR, abs/1312.6199, 2014b*, 2014. URL http://arxiv.org/abs/1312.6199.

[5] Ian J. Goodfellow, Jonathan Shlens and Christian Szegedy Explaining and Harnessing Adversarial Examples In *ICLR*, 2015.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fei ImageNet: A Large-Scale Hierarchical Image Database In *CVPR*, 2009.

[7] Papernot, N., McDaniel, P., Jha S., Fredrikson, M., Celik, Z. B., And Swami, A. The limitations of deep learning in adversarial settings. In *IEEE*, 2016.

[8] Nicholas Carlini David Wagner Towards Evaluating the Robustness of Neural Networks In *arXiv preprint arXiv:1608.04644*, 2016.

[9] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, Ananthram Swami The Limitations of Deep Learning in Adversarial Settings In *arXiv:1511.07528*, 2015.

[10] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, Cho-Jui Hsieh ZOO: Zeroth Order Optimization based Black-box Attacks to Deep Neural Networks without Training Substitute Models In *arXiv:1708.03999*, 2017.

[11] Nicholas Carlini and David Wagner. . Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. In *arXiv:1705.07263*, 2017.

[12] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu Towards Deep Learning Models Resistant to Adversarial Attacks. In *arXiv:1706.06083*, 2017.

[13] Shumeet Baluja and Ian Fischer Adversarial Transformation Networks: Learning to Generate Adversarial Examples In *arXiv:1703.09387v1*, 2017.

[14] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, Ananthram Swami Adversarial The Limitations of Deep Learning in Adversarial Settings In *arXiv:1511.07528*, 2015.