Generating images to fool trained neural networks using simple search algorithm

Tomislav Dobrički, Ognjen Vlajić, Dragan Vidaković, Jelena Slivka
Faculty of Technical Sciences, University of Novi Sad
dtoma95@uns.ac.rs, ogi.vlajic@gmail.com, vdragan@uns.ac.rs, slivkaje@uns.ac.rs

***Abstract*— This paper addresses the problem of generating images, called adversarial images, that can "fool" a neural network (NN) trained for image classification. The NN is considered to be fooled if it assigns the generated image with high confidence to one of its classes. In this paper a simple search based solution is presented, with emphasis on the fact that detailed knowledge about the NN's internal structure and the type of data it was trained on is not needed. This highlights the security vulnerabilities NNs have in regards to the simplicity of algorithms that can fool them. In addition to fooling neural networks, the developed solution can be used for dataset augmentation with the goal of achieving better generalizations of the NN model. Also, the solution may be used for understanding complex NN models. The solution was verified by using NNs of different architectures and trained for different classification tasks, and it performed well in all cases.**

***Keywords*—Artificial neural network, adversarial data, convolutional neural network, search algorithm, image classification, machine learning**

I. INTRODUCTION

An algorithm that can generate images that fool a specific neural network (NN), which was trained for image classification, can potentially be used for several purposes. First of all, it can be used when evaluating a NN performance when it is presented with artificial examples. With this we can discover security vulnerabilities that might be present in system that use NNs for image classification. The generated images can also be used for additional training of a neural network, and thus achieving better generalization [1]. Also, the algorithm can prove helpful for better understanding the way a neural network learned to classify images.

The goal of this paper is to present a simple and easy to implement algorithm that generates images that can fool a specific NN into classifying the image as one of its classes. Several solutions for this problem already exist, one example would be the solution presented in *Explaining and harnessing adversarial examples* [1]. Another example is the generative adversarial network [2] that uses adversarial data generation to train neural networks more effectively [3]. The solution presented in this paper is less complex than the previous examples and shows that fooling a neural network does not require a complicated approach. Also, the described algorithm does not require detailed knowledge of the internal structure of the neural network in question (weights, number of layers, activation functions, etc.), while the other approaches mostly rely on this information.

To verify the effectiveness of the search algorithm presented in this paper, three different NNs trained for image classification were used. The algorithm is tested by seeing if it is able to generate images that can fool the given NN, for each class that the NN trained to recognize. Two of the NNs used for verification are trained for handwritten digit recognition. One of them is a fully connected NN while the other is a convolutional neural network. The third NN is trained for face detection, and is also a fully connected network. The algorithm was able to successfully generate images in all of the cases that were tested.

This paper is structured in the following way. Section II a paper that covers a similar topic is presented. Section III contains the description of two approaches that were developed. The first approach, direct approach, is not considered as a successful solution, but it serves as a basis for the second approach, search based approach. Section IV contains the verification results and description of the NNs used to verify the algorithm. And in the end, the conclusion of this paper is presented in section V, together with suggestions for further work on this topic.

II. PREVIOUS WORK

A paper that covers the topic of generating images to fool neural networks is called *Explaining and harnessing adversarial examples* [1]. The authors of this paper analyze the reason why data that can fool neural networks appear. The paper highlights the NN's linear nature as one of its main weaknesses. Additionally, they mention that NNs that are trained on datasets from the real world often incorrectly classify (or get fooled by) data that is artificially generated. The paper also showcases methods with which adversarial data (data that fools a neural network) can be used to improve the training process of NN classifiers.

The same paper describes a process for generating adversarial images by altering already existing images, which belong to one class, so that the NN recognizes it as a completely different class. This is achieved by changing the pixel values of the image by values that are small enough to be unnoticeable to humans, but manage to fool a NN classifier. One example of this is given on figure 1, where an image of a panda is altered so that the NN classifies it as a gibbon.

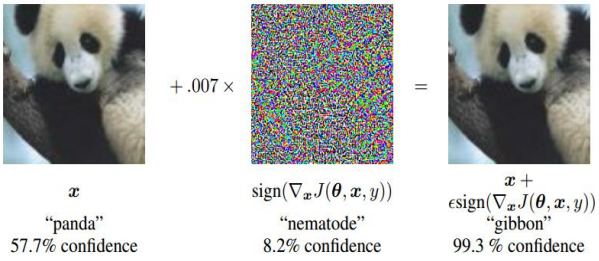


Figure 1. Example of fooling a Neural Network. This image was taken from *Explaining and harnessing adversarial examples* [1].

The search algorithm presented in our paper does not generate adversarial images from already existing images; it creates them on its own. This means that the algorithm does not require any prior knowledge about the types of images that are being classified. In addition, it does rely on knowledge about the internal structure or the weight values in the network, so the algorithm could theoretically be applied to any NN.

III. Methodology and Solution

In this section two different approaches are presented, the direct approach and the search based approach. The direct approach is inefficient and impractical, because it relies on internal knowledge about the Neural Network, but it represents the mathematical basis for the search based approach.

A. Direct approach

The activation of each neuron in a neural network can be represented with a mathematical formula. For instance, the output value of the neuron a_4, using the symbols from figure 2, would be

$$a_4 = x_1 w_{41} + x_2 w_{42} + x_3 w_{43} + \delta_4. \quad (1)$$

The x variables represent values from the input vector, the w variables are weights in the NN, and δ represents the bias value. For now, activation functions are omitted in favor of simplicity.

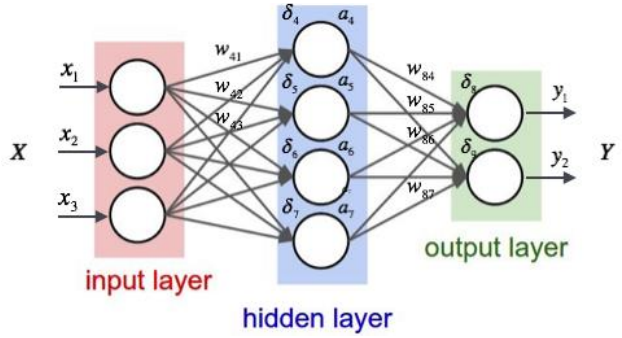


Figure 2. Example of a simple neural network

Under the assumption that the NN from figure 2 is trained for image classification, we can find an image that fools it by finding the values of each i for which the activations of specific neurons in the output layer are maximized. For this purpose we can represent the activation of an output neuron as

$$y_1 = a_4 w_{84} + a_5 w_{85} + a_6 w_{86} + a_7 w_{87}. \quad (2)$$

In this case, we are calculating the optimal values for the input vector; this means that we can ignore the bias value in our calculations because it does not affect the result. By including each value of a in to (2) we get

$$y_1 = K_1 x_1 + K_2 x_2 + K_3 x_3 + K_4 x_4. \quad (3)$$

Each input value x has its own coefficient K that represents a sum of NN weight products. As an example, the value of one of these coefficients would be

$$K_1 = w_{41} w_{84} + w_{51} w_{85} + w_{61} w_{86} + w_{71} w_{87}. \quad (4)$$

A coefficient of x can also be interpreted as the sum of all possible paths from x to the desired output node to the desired output node y in the neural network. A path is represented by the product of all weight values on that path. On figure 3 the paths from (4) are shown as red lines.

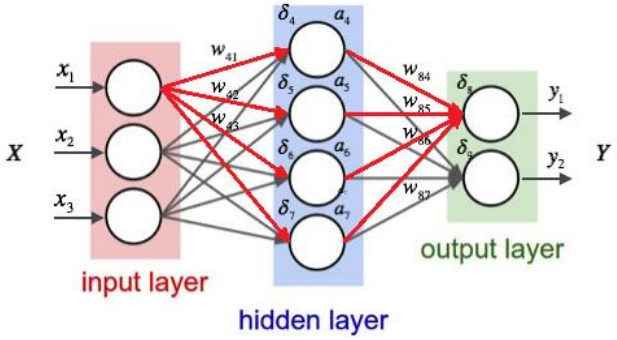


Figure 3. Example of a simple neural network with each path from x_1 to y_1 represented as a red line.

Because the neural network is already trained, the values of its weights are constant. This means that it is possible to calculate the value of the coefficient for each input value x, and find the best value of the input vector. If the coefficient is a number that is greater than zero, x should have as big a value as possible; and in the opposite case, as small as possible. This means that, when the input to the neural network is an 8-bit color image (for example), the values of the generated image could rather be 0 or 255.

The direct approach is conceptually simple because many details regarding NNs are ignored in its explanation. Most notably activation functions, convolutional layers and pooling layers, which are used often when solving image classification problems. In addition, to successfully implement this approach knowledge of the exact weight values of the NN is needed, which is something we wanted to avoid when developing our solution. Because of these reasons, the direct approach can be a failed attempt at solving our problem. Nevertheless, the conclusions gained from developing the direct approach proved useful further on.

B. Search based approach

Generating an image to fool a NN can be represented as search problem, where the goal is to find a good enough combination of pixel values in the set of all possible combinations for a given image size. The space of all possible images would be 256^n where n is the total number of pixel values in the image, and 256 represents all the possible values an image can have in a standard 8-bit color image. Thanks to the conclusions from our direct approach, this space can be reduced to only 2^n. Instead of all possible pixel values, we can take into account only the maximal and the minimal possible ones (0 and 255).

The search algorithm used is a very simple brute force-like algorithm that uses the neural network as a heuristic function. The algorithm starts from an initial image (usually a zero image) and alternates pixel values between 0 and 255 one by one. For each value changed the current image state is given to the NN to predict. If the NN returns

an improved confidence for the desired class, then the change is kept, otherwise it is reverted. This process is repeated until the NN returns a good enough value. For a fully connected NN a whole iteration through every pixel of the image is enough. For convolutional neural networks more iterations are necessary for a result to be found. The number of iterations depends on the size of the input vector and the complexity of the network, but can also vary for different classes of the same network. In the cases that were tested, the number of iterations was always less than 15.

IV. Verification

The algorithm presented in this paper was evaluated using three different neural networks. During the verification process the confidence with which the NN classifies the generated image is observed to be very high in all cases. The confidence is often higher than it is when classifying images from the dataset the NN in question was trained on.

The first NN used for verification is a fully connect network trained to recognized digits from the MNIST handwritten digits dataset [4]. This NN reached a 98.15% accuracy on the MNIST test set. Our search algorithm successfully generated images that the NN predicted to belong to one of its classes with 100% confidence. Some of the resulting images are displayed in table 1.

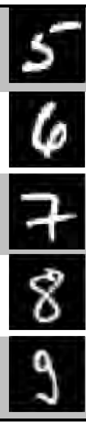
TABLE I.
Some of the images that fooled the fully connected network trained on the MNIST dataset. Image size: 28x28; Grayscale

Generated Image	Example from the same class in the MNIST Dataset
	

It is evident that none of the generated images resemble their target classes. It is also interesting to note that the image that was generated for the digit "1" is a mostly black image with very few white pixels. This might be due to the nature of the MNIST dataset, where "1" is the smallest digit, and therefore uses the smallest number of white pixels. Because of this the NN has learned that images with few white pixels likely belong to the "1" class.

The second network that the algorithm was verified on is a convolutional neural network that was also trained on the MNIST dataset. This network reached 98.64% accuracy when tested. This time the algorithm had to run for around 10 iterations for each class in order to generate good enough images. The resulting images fool the network with around 98% confidence. The resulting images are visible in table 2. This time, some of the generated images do slightly resemble their target classes, most notably for the digits "5" and "9".

TABLE II.
Images that fooled the Convolutional Neural Network trained on the MNIST dataset. Image size: 28x28; Grayscale

Generated image	MNIST dataset example	Generated image	MNIST dataset example
			

The final NN this algorithm was tested on is a fully connected NN trained for face detection. The *Labeled faces in the Wild* [5] dataset and the CIFAR-10 [6] datasets were used for training. While the NN performed well when tested on test sets that are similar to the ones it was trained on (above 90% accuracy), it does not show nearly as good results when presented with real world data. On figure 4 the image that the NN classified as a face with 100% confidence can be seen.

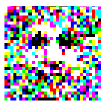


Figure 4. Generated image that fooled the NN for face detection. Image size: 32x32; RGB

V. Conclusion

This paper tackles the problem of generating images to fool neural networks trained for image classification. The presented solution is a simple search algorithm that can fool neural networks trained on different datasets and with different architectures. In addition, the algorithm does not rely on any knowledge about the internal structure of the NN. The algorithm was verified on three different neural networks and each of them classified the generated images with very high confidences.

Although the algorithm was developed to fool image classifying NNs, it can be used for any type of input. This type of algorithm shows that generating fake data that fools NNs is not a complex issue, and that proper security measures, such as hiding the confidence values that the NN returns, should be taken as often as possible.

The algorithm presented in this paper is deterministic in nature because it will always find the same image for any given class. Further work might include developing similar search based algorithms that generate a larger set of images, which could, in turn, be used as additional training data for the NN. The networks used to verify the algorithm are fairly simple compared NNs that are modernly used, so another form of further work would be testing the algorithm on more complex systems.

REFERENCES

[1] Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy. *Explaining and harnessing adversarial examples*. 20 Mar 2015.

[2] Ian J. Goodfellow , Jean Pouget-Abadie , Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair , Aaron Courville, Yoshua Bengio. *Generative Adversarial Nets*. 11 Dec 2014.

[3] Ganin Y, Ustinova E, Ajakan H, Germain P, Larochelle H, Laviolette F, Marchand M, Lempitsky V. *Domain-adversarial training of neural networks*. The Journal of Machine Learning Research. Jan 2016.

[4] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. *Gradient-based learning applied to document recognition*. Proceedings of the IEEE, 86, 2278–2324. 1998.

[5] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. University of Massachusetts, Amherst, Technical Report 07-49, October, 2007.

[6] Alex Krizhevsky, *Learning Multiple Layers of Features from Tiny Image*s, 2009.