Learning universal adversarial permutation functions on images with residual networks

Luca Leeser

December 11, 2017

Abstract

Neural networks have recently seen huge success in practically every domain they have been applied to. In particular, they have revolutionized computer vision and are currently seeing deployment in the real world. Recent research has explored the possibility of tricking these models with adversarial image perturbation with great success. The possibility of tricking neural networks in this fashion raises questions as to their robustness and whether we are at a point where we can apply these models in high-risk domains such as self-driving-vehicles. The general methodology of generating an adversarial image involves selecting an individual image to perturb and a new label to misrepresent with. Gradient descent is then performed on the image to maximize the probability output of the target network with the new label, while minimizing the euclidian distance between the vector representation of the original image and the perturbed image. My research focuses on generalizing this practice by utilizing a neural network architecture that can perturb any input image such that it is misclassified by a target convolutional network.

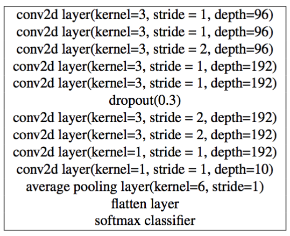
1 Introduction

Adversarial examples for neural networks have been a topic of discussion that brings into question whether or not neural network models can be implemented safely in real-world settings, particularly in ones that have a high impact on the lives of the humans relying on them [6]. Such perturbation algorithms have, until recently, thought to not have been robust in the real world, particularly when image translation and rotation are applied to the perturbed examples. Also, these implementations all have involved selecting an individual adversarial label to trick the network with. However, recent papers have shown that both of these assumptions are not the case, further bringing into question whether they are yet ready for real-world domains. In this paper we explore the possibility of utilizing a neural network to learn a universal perturbation function that can perturb any inputted input to trick the neural network to classify it as something it isn't. I also propose two objective functions that can be utilized to trick the target network. My implementation shows that it is indeed possible to drastically reduce the performance of a convolutional neural network to that of random guessing by utilizing a simple architecture trained on a simple objective function trained to maximize the loss of the target neural network.

2 Approach

2.1 Target Classifier

I first trained a Convolutional Network on the Cifar10 dataset, a dataset of 60,000 training samples and 10,000 testing samples of 32 x 32 color images with classes comprised of animals and vehicles. The network was a deep, fully convolutional network identical to the network described in the paper "One pixel attack for fooling deep neural networks" by JiaWei Su et. al[7]. The network comprised of 9 convolutional layers with dropout in the middle and an average pooling layer at the end, before a flatten and softmax activation were applied. After training, this network achieved a testing accuracy of 75% on cifar10.

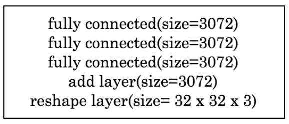


conv2d layer(kernel=3, stride = 1, depth=96)
conv2d layer(kernel=3, stride = 1, depth=96)
conv2d layer(kernel=3, stride = 2, depth=96)
conv2d layer(kernel=3, stride = 1, depth=192)
conv2d layer(kernel=3, stride = 1, depth=192)
dropout(0.3)
conv2d layer(kernel=3, stride = 2, depth=192)
conv2d layer(kernel=3, stride = 2, depth=192)
conv2d layer(kernel=1, stride = 1, depth=192)
conv2d layer(kernel=1, stride = 1, depth=10)
average pooling layer(kernel=6, stride=1)
flatten layer
softmax classifier

Figure 1: Target classifier network architecture

2.2 Adversarial network

The adversarial network that learns the permutation function is a simple four layer residual network with three dense layers and an add layer at the end, this add adds the function learned by the prior layers with the original input layer, thus the network will ideally learn some perturbation function that can simply be applied to the input image.



fully connected(size=3072)
fully connected(size=3072)
fully connected(size=3072)
add layer(size=3072)
reshape layer(size= 32 x 32 x 3)

Figure 2: Adversarial network architecture

3 Objective Functions

In this part, I propose two objective functions that try to maximize on the target classifier's output. The difference between the two is mainly based on the difference metric used between the target classifiers output on an unperturbed datapoint and its permuted counterpart.

3.1 max-norm maximization

Max-norm maximization, is described as follows.

$$\min \frac{1-a}{||f(x) - f(\phi(x))||_\infty} - a||x - \phi(x))||_2$$

Where the function f is the network we are trying to trick, the function is the permutation function of the first three layers, and x is the input image being permuted, lastly a is a hyper-parameter that manages the ratio between the two constraints. This function attempts to maximize the max norm of the softmax output of the target classifier between the original datapoint and its permuted version.

3.2 Cross-entropy maximization

The next objective function, cross-entropy maximization, is described as follows:

$$\min \frac{1-a}{C(f(x), f(\phi(x)))} - a||x - \phi(x))||_2$$

Where the function C is the categorical cross entropy between the output of the target network on the original datapoint and its permuted version.

4 Evaluation

4.1 Setup

I first trained the classifier on Cifar10 for around 100 epochs, at which point I was acheiving a training accuracy of 97.3% and testing accuracy of 75.2%. After training the residual network on both objective functions, I applied the trained network on the entirety of the test set and ran the original classifier on the permuted test set. From Figure 3, you can see that the max-norm maximization objective function was more effective at tricking the the network when compared to cross entropy maximization. However, both methods led to a significant decrease in the classification performance of the original network. The reason why max-norm performed better than cross entropy is likely due to it actually learning a permutation that results in every image being classified as the same class. It also learned a permutation that was more similar to the original image than the permutation found with cross entropy maximization. However, max norm failed to trick the network when the input image had the same class as the class it had learned, thus the cross entropy function still shows some promise when compared to max norm.

Objective Function	Testing Accuracy
Unmodified	75.4%
Max-norm maximization	12.4%
Cross entropy maximization	26.7%

Figure 3: Comparison of testing accuracy of target network after permutation applied to testing set of 10,000 images.

4.2 Importance of hyperparameters

From training on both objective functions, it became immediately apparent how important the alpha hyperparameter was with regards to convergence. If alpha was too low, the network would learn a permutation that was completely unrecognizable to the user. With an alpha that was too high, the weights of the hidden layers would collapse to zero, and the network would just become the identity function. From utilizing random search, I settled on an alpha of around 0.99999, which resulted in a significant reduction of the classifier performance and still produced images that were easily identifiable to a person.

4.3 Max-norm maximization

The residual network trained on the max-norm maximization objective function appeared to learn a very sparse permutation function, as can been seen in the picture comparison. During training, the network seems to select a class at random where it learns a permutation that forces every input to classify as that label. This is likely because this objective function will learn a weight that the classifier might be over-fitting on, and amplify that particular weight drastically, causing every output to classify as that label. This is evidenced by the fact that the permutation learned from max-norm maximization is very sparse, especially in relation to the one learned by cross-entropy maximization. This results in very consistent classification, however can easily be combated by normalizing the input before feeding it through the target network.

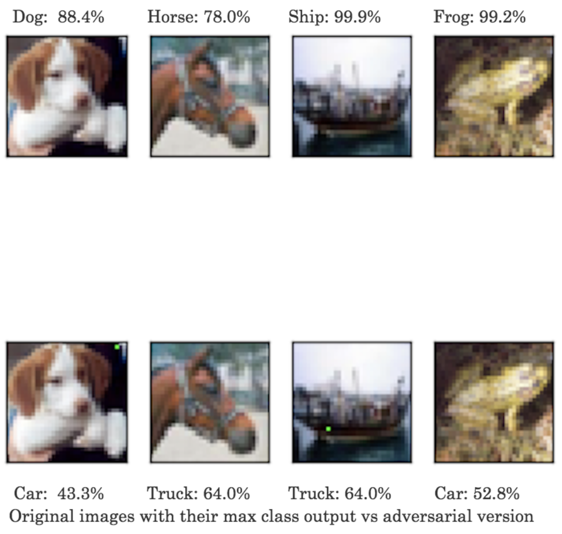


Figure 4: Original Images with their perturbed counterpart after training the residual network with the max-norm maximization objective function. As shown, the permutation function likely learned to amplify weights of high importance for classifying a car or truck.

4.4 Cross-entropy maximization

The residual network trained on the cross entropy maximization function ended up learning a less sparse permutation. Also, unlike with max-norm maximization, the class output was not constant. With the new output changing depending on the permutation, the permutation learned was also not one-to-one, as can be seen in Figure 2. Two of the sample inputs were Trucks, but they had different outputs once permuted. This result is quite interesting and further analysis into the set of learned permutations would be an interesting continuation from these results.

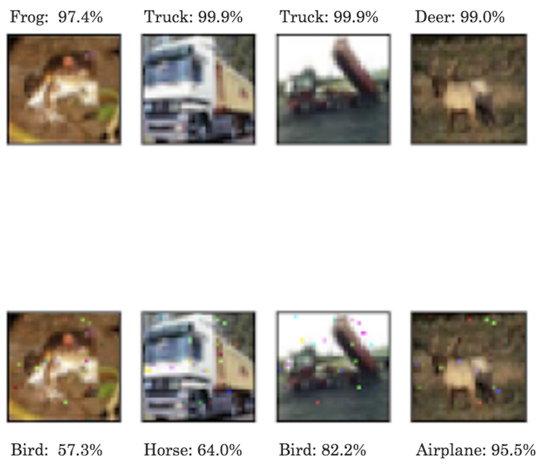


Figure 5: Original Images with their perturbed counterpart after training the residual network with the cross-entropy maximization objective function.

5 Relevant Work

Methods of producing adversarial image perturbation utilizing first-order optimization schema generally optimize based on an l2-constrained loss maximization problem:

$$\max L(f(x), y)\, s.t.\, ||\phi(x)) - x||_2 < e$$

Where L is the loss function, f is classifier you are trying to trick, is your permutation function and y is the ground truth label for the datapoint x. Several well known and simple methods include Fast Gradient Vector [1], DeepFool [2], as well as the Universal Perturbations method [3]. Fast Gradient Vector is a simple one-step gradient ascent method that selects the softmax log-loss function as L in the prior equation, and performs one-step gradient descent on a target label for some input x, which finds a permutation that maximizes the loss of the classifier while keeping the image within a set l2 distance from the original. Unlike FGV, DeepFool optimizes on the objective:

$$\min ||\phi(x) - x||_2\, s.t.\, argmax f(\phi(x)) \neq y$$

Which instead finds a minimum perturbation function on x such that the output of the classifier is wrong. The universal perturbations method is just a generalization of equation 1, where the perturbation function and loss function are shared over the entire dataset. Although their objective was similar to mine in finding a universal permutation, I ultimately wanted was to see if it was possible to implement a neural network that could learn such a universal perturbation function.

Another recently published paper, Synthesizing Robust Adversarial Examples [4] argues that the original objective function performs poorly due to the fact that the adversarial examples perform poorly when further permutations, which are inevitable in a real-world context, are further applied. Instead, they propose a new objective function, Expectation over transformation:

$\hat{x} = argmax E_{t \in T}(\log P(y|t(x'))s.t. E_{t \in T}(d(t(x') - t(x))) < e$

This objective function instead optimizes over a chosen distribution of permutation functions (t), this distribution t can include distortions including noise, rotation and translation. These are all permutations that would break the permuted images optimized on the first objective function. The result is a very robust method for implementing adversarial examples that can still trick the classifier regardless of angle or rotation. They prove this by generating an adversarial 3d representation of a turtle as shown in figure 6.

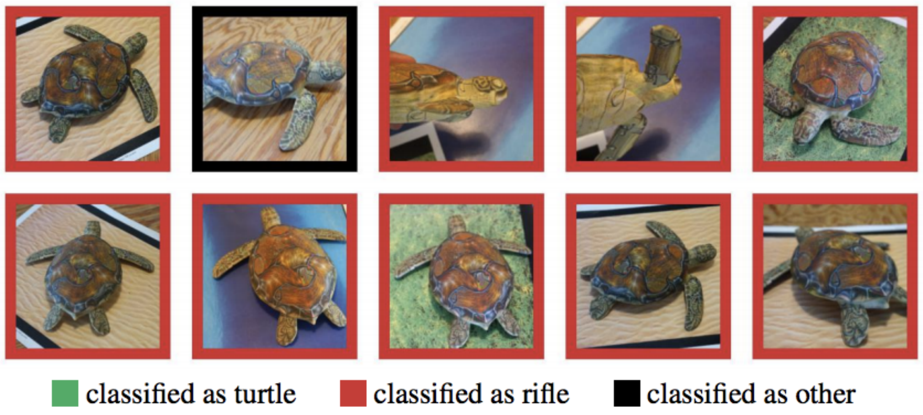


Figure 6: Image example from the paper Synthesizing Robust Adversarial Examples by Anish Athalye et al. "Randomly sampled poses of a single 3D-printed turtle adversarially perturbed to classify as a rifle at every viewpoint by an ImageNet classifier." [4]

6 Optimizations from the course

I utilized several optimization methods and techniques that we learned from the course. Firstly, I used Adam optimizer as my optimization method when training the classifier and the adversarial network. I also trained on a cpu and a Titan Xp GPU to compare the wallclock times. I also utilized early stopping while training, this was helpful because the network would sometimes collapse into the identity function so I was able to select the model with lowest validation error before it had collapsed. The wallclock analysis between CPU and GPU training times can be seen in the following table.

Hardware	Classifier	Adversarial
AMD Ryzen 7 1700 w/ 16 threads @ 3.9 Ghz	2:43:29	0:3:10
Nvidia Titan Xp	0:20:15	0:0:20

7 Further Research

These results are quite promising and there are several possible paths that this project could continue with. It would be interesting to see if the network could also learn a robust perturbation function using an objective function similar to that of the one described in the paper, Synthesizing Robust Adversarial Examples. Also, it would be useful to perform the same experiment on more state-of-the-art classifiers such as GoogleNet.

References

[1] A. Rozsa, E. M. Rudd, and T. E. Boult. Adversarial diversity and hard positive

generation. In CVPRW, 2016.

[2] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In CVPR, 2016.

[3] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. In CVPR, 2017.

[4] Anish Athalye, L. Engstrom, A. Ilyas, K. Kwok. Synthesizing Robust Adversarial Examples. Under review at ICLR, 2018.

[5] JiaWei Su, Danilo Vasconcellos Vargas, S. Kouchi. One pixel attack for fooling deep neural networks

[6] M. Barreno, B. Nelson, A. D. Joseph, and J. Tygar. The security of machine learning. Machine Learning, 81(2):121– 148, 2010.

[7] J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In ICLR (workshop track)