Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 3, 2021, Pages. 8397 - 8404
Received 16 February 2021; Accepted 08 March 2021.

One Pixel Attack Analysis Using Activation Maps

Shubham Sinha[1], S.S. Saranya[2]

[1]Final Year Student, Computer Science Engineering, SRM Institute of Science and Technology, Kattankulathur Campus, Tamil Nadu, India. E-mail: sr7006@srmist.edu.in

[2]Assistant Professor, Department of Computer Science and Engineering, School of Computing, SRM Institute of Science and Technology, Kattankulathur Campus, Tamil Nadu, India. E-mail: saranyas6@srmist.edu.in

ABSTRACT

According to researchers, when a small amount of perturbation (a small change) is added to the input of a Deep Neural Network (DNN), the output of the DNN can be altered easily. In this paper, we'll be performing this attack by perturbing one pixel of the input image for the image classifying model. We'll also take a look at the activation maps that will help us to visualize the working of the attack as well as classification of the image. To perform the attack we'll be using an optimization algorithm called Differential Evolution (DE) which is going to help us to generate adversarial perturbation for one pixel. This is a grey box attack, meaning that we don't have much information about the target model, and this attack can fool many types of DNN because of the features provided by DE. The results shows that images present in Dataset of Kaggle (CIFAR-10) and ImageNet Dataset can be attacked just by modifying one pixel of the image. This exact vulnerability exists in original CIFAR-10 Dataset. Thus, this attack shows that currently present Deep Neural Networks aresusceptible to such low dimension attacks.

KEYWORDS

DNN, DE, Pixel, Vector, Dataset, Vulnerability, Attack.

Introduction

Researchers have shown already that Deep Neural Networks (DNNs) aren't that secure and they're vulnerable to various adversarial attacks. But what causes these vulnerabilities is still not known; research is still going on to find the cause of this vulnerability. Many experiments have shown that by adding perturbation in the input image vector can make the image classifying model to mis-classify the image. Since then, many algorithms have been created to form this "adversarial image". The general idea behind the adversarial image formation is just to add little amount of perturbation in the input image.

So this pixel modification might or might not be visible to the human eye but it won't change how humans recognize the images but this modification can change the image classifier's output. But the perturbation vector should not be large, meaning that one should not modify too many pixels in the image otherwise the modification will be visible to the human eye as well.

So in this project, we'll be performing the attack only on one pixel and the only thing which we need to know is the probability labels of the image classifier for normal images.

The advantages of the proposed attack are as follows:

1. Effective - Three different neural networks on Kaggle's CIFAR-10 Dataset gave around 68%, 71%, and 63% success rates and 22%, 35%, and 31% success rates on original CIFAR-10 Dataset.
2. Grey Box Attack - One doesn't need any information about the internals of the model but one needs the probability labels for the target classes.
3. Flexibility - This attack works fine with almost all types of DNN models. Doesn't matter how complex the model is or how much training it has undergone.

At the end of the paper, we'll be looking at locality analysis. In there the attack will be performed near to the pixel on which the attack was successful, same perturbation vector will be used but just with a different pixel. It was shown that the success rate of nearby pixels is kind of same with the actual attacked pixel. This analysis actually shows that it is not the pixels or neurons that is vulnerable, it is the receptive fields. This is what make this attack independent of the model.

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 3, 2021, Pages. 8397 - 8404
Received 16 February 2021; Accepted 08 March 2021.

Methodology

A. Activation Maps for One Pixel Attack

To understand how one pixel attack works, we could use Propagation Maps that will allow us to visualize the perturbations of every layer in the neural network. Figure 1 shows the activation maps for the classification of a bird on Resnet Model. It can be seen that DNN tries to find and extract features from the images at every layer of the DNN. However this is just a normal classification, the input vector wasn't perturbed. The activation maps changes when the input vector is perturbed.

B. Single Pixel Perturbations that can Change Class

In Figure 2 we can see that the perturbation started small and then started to spread in further layers, in the last layer we can see that the perturbation has left some residuals exactly at the place of the perturbation coordinates. All of the adversarial samples actually shared similar features of the PMaps.

So one pixel change can cause influences that spreads over the entire map, especially in deeper layers.

C. Single Pixel Perturbation that DO NOT Change Class

Let us say, we picked the wrong pixel to attack, in that case we can see from Figure 3 that the influence's density decreases layer-by-layer and our attack failed. The image was classified correctly.

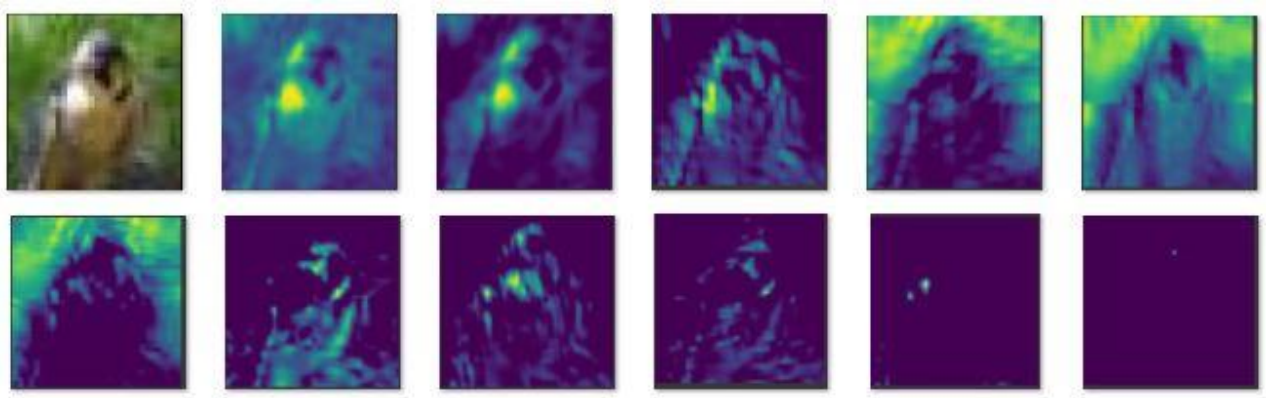

Figure 1.PMap for Resnet Model. Without any perturbation, this image was correctly classified as a bird

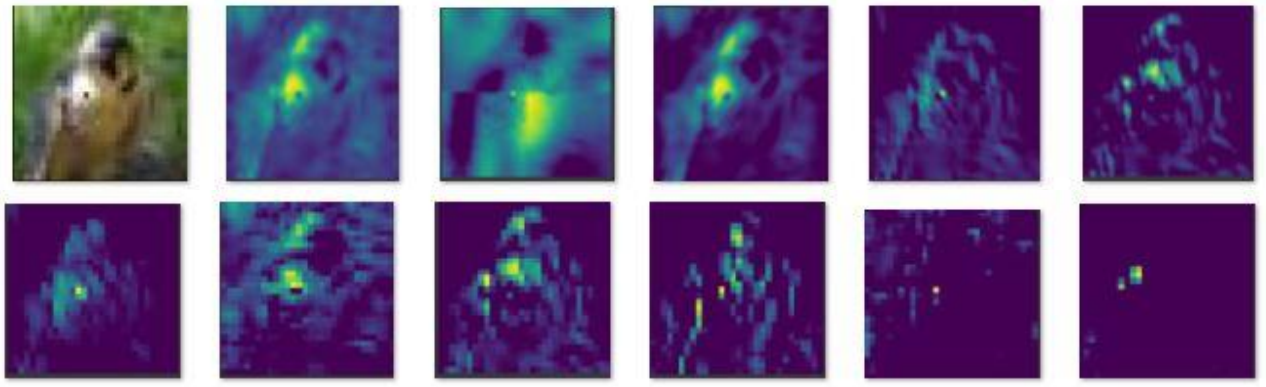

Figure 2.PMap for Resnet Model. The bird was misclassified as frog after one pixel change

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 3, 2021, Pages. 8397 - 8404
Received 16 February 2021; Accepted 08 March 2021.

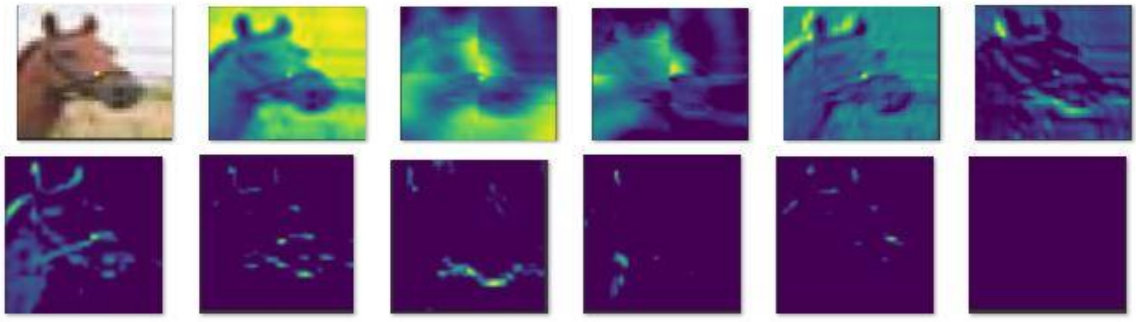

Figure 3. Perturbation failed to change the class. The image was successfully classified as a horse

However, this is not a rule, in the Figure 4, we notice that the pixel's influence is strong enough but still the image was classified correctly.

So we can say that it is not necessary that high influence of the perturbation might give us successful attack, it strongly depends on the position of the pixel and sample.

D. Activation Maps Generation

Activation maps can be generated using a python modules named keract. keract can be used to generate the activation maps or even heat maps for each layer of the DNN model. To get the above images, one needs to use the get_activations and display_activations functions.

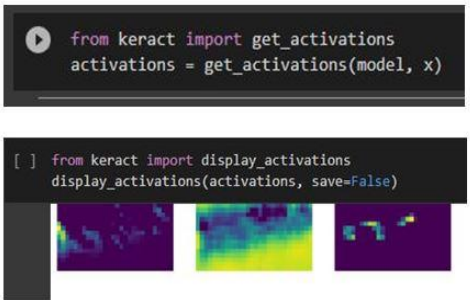

Figure 4. Keractactivation maps generation in Jupyter Notebook
Just like that we can have heat maps as well.

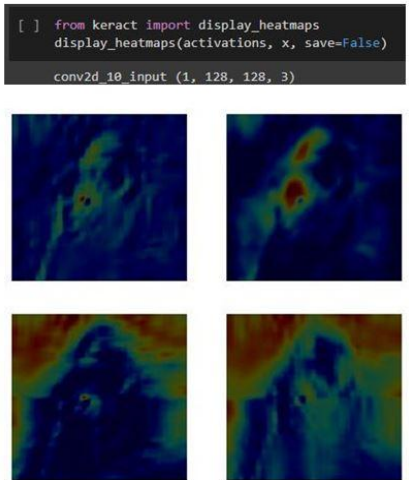

Figure 5. Keractheat maps generation in Jupyter Notebook

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 3, 2021, Pages. 8397 - 8404
Received 16 February 2021; Accepted 08 March 2021.

One Pixel Attack

The DNN model being used is ResNet and the model is around 92% accurate. The dataset being used is CIFAR-10.

So the following image was successfully classified as a bird with an accuracy of ~70%.

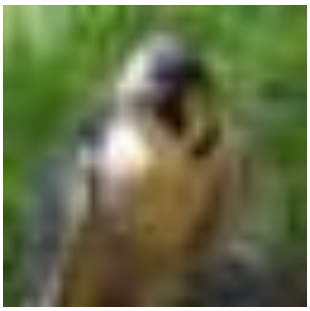

Figure 6. Bird (32x32)

Now the function perturb_image(xs, img) will take perturbation vector (xs) and the image (img) as input and perturb the image as per the perturbation vector and return the perturbed image (Figure 7).

```
if xs.ndim < 2:
    xs = np.array([xs])

# Copy the image n == len(xs) times so that we can
# create n new perturbed images
tile = [len(xs)] + [1]*(xs.ndim+1)
imgs = np.tile(img, tile)

# Make sure to floor the members of xs as int types
xs = xs.astype(int)

for x,img in zip(xs, imgs):
    # Split x into an array of 5-tuples (perturbation pixels)
    # i.e., [[x,y,r,g,b], ...]
    pixels = np.split(x, len(x) // 5)
    for pixel in pixels:
        # At each pixel's x,y position, assign its rgb value
        x_pos, y_pos, *rgb = pixel
        img[x_pos, y_pos] = rgb

return imgs
```

Figure 7.perturb_image(xs, img)

This is what the perturbed image looks like after this

Figure 8. Bird (Perturbed) (32x32)

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 3, 2021, Pages. 8397 - 8404
Received 16 February 2021; Accepted 08 March 2021.

Now passing this perturbed image to the classifier, this was the obtained result.

```
image_id = 384
pixel = np.array([16, 13,  25, 48, 156])
model = resnet

true_class = y_test[image_id, 0]
prior_confidence = model.predict_one(x_test[image_id])[true_class]
confidence = predict_classes(pixel, x_test[image_id], true_class, model)[0]

print('Confidence in true class', class_names[true_class], 'is', confidence)
print('Prior confidence was', prior_confidence)
helper.plot_image(perturb_image(pixel, x_test[image_id])[0])
```

```
Confidence in true class bird is 0.00018887517
Prior confidence was 0.7066183
```

Figure 9. Output for Perturbed Image

So the DNN was not able to identify this image as a bird. The confidence of this image being a bird is 0.00018887517 but previously this image was classified as a bird with an accuracy of 0.7066183.

How this modified one pixel affects the classification has been explained in the previous section where activation maps were being used to show the output of each layer of DNN. But here the differential evolution wasn't used anywhere. This was a random attack which somehow got succeeded but we cannot guarantee that for every random perturbation vector this will work.

The differential evolution is already available in scipy module ofpython but we need to make some changes so that it could work with the complete tuple of perturbation vector, i.e. (x, y, r, g, b) and measure the optimization on the prediction function of the model, plus we've to change the callbacks as well. Start the attack with a random perturbation vector then feed differential evolution with these arguments then collect the results.

So we need a function to check whether the perturbation vector was a good one or not. The following function will do the check for us.

```
def attack_success(x, img, target_class, model, targeted_attack=False, verbose=False):
    # Perturb the image with the given pixel(s) and get the prediction of the model
    attack_image = perturb_image(x, img)

    confidence = model.predict(attack_image)[0]
    predicted_class = np.argmax(confidence)

    # If the prediction is what we want (misclassification or
    # targeted classification), return True
    if verbose:
        print('Confidence:', confidence[target_class])
    if ((targeted_attack and predicted_class == target_class) or
        (not targeted_attack and predicted_class != target_class)):
        return True
    # NOTE: return None otherwise (not False), due to how Scipy handles its callback function
```

Figure 10. Success Criterion Function

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 3, 2021, Pages. 8397 - 8404
Received 16 February 2021; Accepted 08 March 2021.

```
    targeted_attack = target is not None
    target_class = target if targeted_attack else y_test[img_id, 0]

    # Define bounds for a flat vector of x,y,r,g,b values
    # For more pixels, repeat this layout
    bounds = [(0,32), (0,32), (0,256), (0,256), (0,256)] * pixel_count

    # Population multiplier, in terms of the size of the perturbation vector x
    popmul = max(1, popsize // len(bounds))

    # Format the predict/callback functions for the differential evolution algorithm
    def predict_fn(xs):
        return predict_classes(xs, x_test[img_id], target_class,
                               model, target is None)

    def callback_fn(x, convergence):
        return attack_success(x, x_test[img_id], target_class,
                              model, targeted_attack, verbose)

    # Call Scipy's Implementation of Differential Evolution
    attack_result = differential_evolution(
        predict_fn, bounds, maxiter=maxiter, popsize=popmul,
        recombination=1, atol=-1, callback=callback_fn, polish=False)

    # Calculate some useful statistics to return from this function
    attack_image = perturb_image(attack_result.x, x_test[img_id])[0]
    prior_probs = model.predict_one(x_test[img_id])
    predicted_probs = model.predict_one(attack_image)
    predicted_class = np.argmax(predicted_probs)
    actual_class = y_test[img_id, 0]
    success = predicted_class != actual_class
    cdiff = prior_probs[actual_class] - predicted_probs[actual_class]

    # Show the best attempt at a solution (successful or not)
    helper.plot_image(attack_image, actual_class, class_names, predicted_class)

    return [model.name, pixel_count, img_id, actual_class, predicted_class, success, cdiff, prior_probs, predicted_probs, attack_
result.x]
```

Figure 11. Function to Perform Attack (Both targeted and non-targeted)

Function shown in Figure 11 allows us to perform both targeted and non-targeted attack and involving differential_evolution to perform the perfect attack on the images.
However still using differential evolution doesn't gurantees that attack will be a success always. Many times differential evolution might fail to find good pixel to perturb the image.

So we could use pandas library to plot the statistics for different models on the same attack.

	model	accuracy	pixels	attack_success_rate
0	lenet	0.7488	1	0.344444
1	lenet	0.7488	3	0.644444
2	lenet	0.7488	5	0.644444
3	pure_cnn	0.8877	1	0.066667
4	pure_cnn	0.8877	3	0.133333
5	pure_cnn	0.8877	5	0.188889
6	net_in_net	0.9074	1	0.100000
7	net_in_net	0.9074	3	0.244444
8	net_in_net	0.9074	5	0.311111
9	resnet	0.9231	1	0.144444
10	resnet	0.9231	3	0.211111
11	resnet	0.9231	5	0.222222
12	densenet	0.9467	1	0.044444
13	densenet	0.9467	3	0.233333

Figure 12. Attack Success Statistics

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 3, 2021, Pages. 8397 - 8404
Received 16 February 2021; Accepted 08 March 2021.

Position Sensitivity and Locality

Till now we've seen that how one pixel attack works just by changing one pixel in the image and it works by searching for that pixel but again, to what extent is the success dependent on the pixel position?

To understand this, let us choose a random pixel to attack and perturb that pixel.
The results are shown in Table 1, which actually show that success rate in case of random pixel is very low. This shows that pixel position plays a vital role when it comes to attack success. However attacking on the near-by pixel (8 Pixels) shows a positive result.

Table 1. Success Rate of the Attack

	LeNet	**ResNet**
1 Pixel Attack (Original)	58%	21%
Random Pixel Attack	4.8%	3%
Nearby Pixel Attack	33%	31.3%

Attacks and Defences

The security problem of Neural Networks became a critical topic after all these attacks.

C. Szegedy et al. was responsible for revealing the sensitivity to well tuned artificial perturbation which we can craft using several algorithms based on gradient. For example "Fast Gradient Sign" algorithm is used for calculating effective perturbation. Other algorithm uses Greedy Approach, Jacobian Matrices and so. Apart from perturbation, there are other ways to create adversarial images that model will misclassify, such as an artificial image, rotating the image and so. The adversarial perturbation is not limited to images only, this can be used in NLP, Speech Recognition, Malware Classification etc.

So many detection and defense methods were proposed to patch the vulnerability, for example, network distillation enhanced the robust nature of the neural network, adversarial training was proposed in which adversarial images are the part of training data. Even few image processing techniques proved to be effective when it came to detection of adversarial images. But again, these defense techniques can be bypassed easily by making a small modification in original attacks.

Conclusions

So in this paper, the following things were highlighted.

- *The influence of the perturbation*

Using the PMaps and Heatmaps we can see how perturbation can influence layers of DNNs. From the figures we can see that perturbation's influence grows and spreads as we go deeper in DNN.

- *Performing the attack on CIFAR-10 Dataset*

The codes shown in Jupyter notebook shows how the attack can be performed on CIFAR-10 Dataset.

- *Measuring the Success Statistics*

Using the pandas library and DataFrame module we generated the table showing the attack success rate on different types of models.

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 3, 2021, Pages. 8397 - 8404
Received 16 February 2021; Accepted 08 March 2021.

Results

In this paper we implemented the functions that were needed to perform the attack, we also discussed about Activation Maps which indeed helped us to understand the working of the attack and DNN, apart from that, the attack was performed on different DNN models and their success rate was shown in the Attack Success Statistics.

References

[1] Image Credits: ImageNet, and CIFAR-10.
[2] Su, J., Vargas, D.V., & Sakurai, K. (2019). One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, *23*(5), 828-841.
[3] Vargas, D.V., &Su, J. (2019). Understanding the one-pixel attack: Propagation maps and locality analysis. *arXiv preprint arXiv:1902.02947*.
[4] Khan, U., Woods, W., &Teuscher, C. (2019). Exploring and Expanding the One-Pixel Attack.
[5] Athalye, A., Engstrom, L., Ilyas, A., & Kwok, K. (2018). Synthesizing robust adversarial examples. *In International conference on machine learning*, *PMLR*, 284-293.
[6] Athalye, A., Carlini, N., & Wagner, D. (2018). Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *In International Conference on Machine Learning*, *PMLR*, 274-283.
[7] Barreno, M., Nelson, B., Sears, R., Joseph, A.D., &Tygar, J.D. (2006). Can machine learning be secure?.*In Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, 16-25.
[8] Symposium on Information, computer and communications security, 16–25, 2006.
[9] Barreno, M., Nelson, B., Joseph, A.D., &Tygar, J.D. (2010). The security of machine learning. *Machine Learning*, *81*(2), 121-148.
[10] Brown, T.B., Mané, D., Roy, A., Abadi, M., & Gilmer, J. (2017). Adversarial patch. *arXiv preprint arXiv:1712.09665*.
[11] Buckman, J., Roy, A., Raffel, C., & Goodfellow, I. (2018). Thermometer encoding: One hot way to resist adversarial examples. In *International Conference on Learning Representations*.
[12] Carlini, N., & Wagner, D. (2017). Adversarial examples are not easily detected: Bypassing ten detection methods. *In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 3-14.
[13] Carlini, N., & Wagner, D. (2017). Magnet and" efficient defenses against adversarial attacks" are not robust to adversarial examples. *arXiv preprint arXiv:1711.08478*.
[14] Carlini, N., & Wagner, D. (2017). Towards evaluating the robustness of neural networks. *In IEEE symposium on security and privacy (sp)*, 39-57.
[15] Dang, H., Huang, Y., & Chang, E.C. (2017). Evading classifiers by morphing in the dark. *In Proceedings of the ACM SIGSAC conference on computer and communications security*, 119-133.
[16] Dhillon, G.S., Azizzadenesheli, K., Lipton, Z.C., Bernstein, J., Kossaifi, J., Khanna, A., &Anandkumar, A. (2018). Stochastic activation pruning for robust adversarial defense. *arXiv preprint arXiv:1803.01442*.
[17] Goodfellow, I.J., Shlens, J., &Szegedy, C. (2014a). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.
[18] Goodfellow, I.J., Shlens, J., &Szegedy, C. (2014b). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.
[19] Grosse, K., Manoharan, P., Papernot, N., Backes, M., & McDaniel, P. (2017). On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280*.
[20] Guo, C., Rana, M., Cisse, M., & Van Der Maaten, L. (2017). Countering adversarial images using input transformations. *In ICLR.arXiv preprint arXiv:1711.00117*.