How to create adversarial examples and mitigate adversarial effects

Junsu Lim[1]

[1] DGIST, Republic of Korea

Abstract.

Adversarial examples have a malicious effect on AI. Seeing influence on adversarial examples at the laboratory level will not be a problem. However, if adversarial examples are applied to AI systems used in real life, personal and property damage may occur. Therefore, in this project, adversarial examples are created and two mitigating methods are presented to mitigate the adversarial effect of adversarial examples. Randomization and autoencoder were used as the mitigating method, the autoencoder showed good results. Specifically, when the image to which gaussian noise was applied was classified by CNN, the accuracy fell to 17.4%, and applying the autoencoder could increase the classification accuracy of the image to 39.6%. The most striking result is that the classification accuracy of adversarial images to which the fast gradient sign method (FGSM) was applied decreased significantly to 6%, but the classification accuracy was increased to 31.7% when autoencoder was applied.

Keywords: Adversarial examples, Fast Gradient Sign Method(FGSM), Gaussian Noise, Randomization, Autoencoder, Transfer learning, Convolution Neural Network

1 Introduction

Due to the development of computing power and semiconductors, deep learning is being used everywhere in our daily lives. It is used in many fields such as voice recognition, image recognition, recommended algorithms and etc. Video and image recognition have made a lot of progress in almost 10 years. Because a model called CNN was developed, and by improving CNN, we were able to develop AI that was better at recognizing images than humans. In particular, the development of CNN has greatly contributed to the development of self-driving cars. Self-driving cars determine whether an object is a person or a car by judging photos taken with a camera in real time. CNN's accuracy is therefore a very important factor for self-driving cars. However, techniques have been developed to attack CNN since 2010, and "Explaining and Harnessing Adversarial Examples" announced by Google in 2015 shows a method for attacking the classification of CNN. This paper makes a great contribution to the field of AI security.

Google paper argued that the cause of existing adversarial effects was not due to the non-linearity property in the activation function, adversarial effects occur in high dimensional linearity than non-linearity[3]. Such arguments are still referenced to the creation of adversarial attacks to this day. In this project, I will design experimental models to determine the impact of adversarial examples on normal CNN models and see how they affect them. Two methods were used to generate adversarial examples. I developed the Gaussian noise and Fast Gradient Sign Method(FGSM) to generate adversarial examples, It also developed a randomization method and autoencoder to mitigate adversarial effects caused by adversarial examples. The method used to mitigate existing adversarial effects were to manipulate CNN. The method used in this project is to mitigate adversarial effects with a simple pre-processing layer without manipulating CNN.

2 Related Works

Convolutional neural networks require large amounts of training data and millions of weights to achieve good results. Training these networks is therefore extremely computationally intensive[1], often requiring weeks of time on many CPUs and GPUs. Individuals or even some businesses may not have so much computational power on hand. The computational burden of training a deep network is addressed through outsourced training. Typically, people use a MLaaS(Machine Learning as a Service) operated by cloud-based system. But when an unreliable third-party cloud provider learns the model we commissioned; it can insert a backdoor. The DNN with backdoor shows other label by a malicious trigger or degrades the performance of the overall model. It can also be seen passing the validation test process. Because the entire feature of the image does not change, and only a specific trigger is inserted into the image.

CNN shows very high performance in various tasks. But it shows that it is extremely vulnerable to adversarial examples. For example, imperceptible perturbations added to clean images can cause CNN to fail. Thus, many studies are being conducted to compensate for these failures, and in this paper, randomization operations are used. Randomization operations consist of random resizing and random padding. Random resizing resizes the input images to a random size, and random padding pads zeros around the input images in a random method. This paper tells four things about the advantages of randomization. 1) Randomization at inference time makes the network much more robust to adversarial examples, but hardly hurts the performance on clean images. 2) There is no additional training or fine-tuning required which is easy for implementation. 3) Very few computations are required by adding the two randomization layers, thus there is nearly no run time increase. 4) Randomization layers are compatible to different network structures [2].

In previous research, it was thought that adversarial effects were caused by non-linearity and overfitting in the neural networks. However, this paper argue that linear nature is a primary cause. They also say that using linear nature makes it easy to create adversarial examples. Because linear behavior in high-dimensional spaces is sufficient to cause adversarial examples. The following expression produces adversarial examples.

$$W^T\tilde{x} = W^T x + W^T\eta \ (\|\eta\|_\infty < \varepsilon, \eta = sign(W))$$

In other words, through formula, if sufficient dimensions are present, even simple linear models can make a large output activation value.

More nonlinear models such as sigmoid networks are carefully tuned to spend most of their time in the non-saturating (where the graph converges at 0 and 1 in sigmoid function), more linear regime for the same reason. The following expression is a non-linear model that produces a perturbation.

$$\eta = \varepsilon sign\left(\nabla_x, J(\theta, x, y)\right)$$

As with the expression above, the neural network model has a linear characteristic, so it is possible to sufficiently change the activation value by differentiating specific x only once in the classification model and moving as much as ε for each axis. This expression creates Fast Gradient Sign Method(FGSM) attack techniques[3].

3 Key Contribution of Your Project

1. Development of high-performance CNN that can receive images of variable size.
2. transfer learning using pre-trained made by step 1.
3. Development of Gaussian noise and Fast Gradient Sign Method(FGSM) methods to generate adversarial examples.
4. Development of mitigating methods to mitigate adversarial effects. Using randomization methods and autoencoder. The key contribution to this project is to check whether the adversarial examples created in step 3 are alleviated by the mitigating methods created in step 4.

4 Methods

Development of high-performance CNN that can receive images of variable size :

The dataset used in this project is CIFAR10. Imported train data was preprocessed using data scaling techniques. I used StandardScaler, which changed the mean to zero and the standard deviation to one. The y-values, which mean label, used the np_utils.to_categorical function for one-hot encoding. When initially designing CNN model, I designed them in the following order.

1 layer : Conv(filters 64) → Conv(filters 64) → pooling → dropout

2 layer : Conv(filters 128) → Conv(filters 128) → pooling → dropout

3 layer : Conv(filters 256) → Conv(filters 256) → pooling → dropout

4 layer : Flatten → Dense(neurons 512) → dropout → Dense(neurons 512) → dropout → Dense(neurons 10)

However, If I used it to learn the model above, the accuracy is in the early 90% and the validation accuracy is in the 70%. The low validation accuracy indicated that overfitting occurred during backpropagation in the training set. I used four techniques to solve this problem. The first is the data augmentation technique. In Keras, ImageDataGenerator() exists for image pre-processing. The function increases the number of images by manipulating the original image. I will not write down the settings applied to the function in detail here. When data augmentation was used only, the validation accuracy was between 82 to 85%. I was not satisfied with this accuracy, so I studied for additional information and knew to add three more things to the model. I add the kernel initializer, batch normalization and then I has done hyperparameter tuning about dropout value.

The reason for using kernel initializer is that unless the weight initialization method is set separately, the Keras layer's weight initialization method is random_uniform. There is a high risk of falling into the vanishing gradient problem or the exploding gradient problem during the backpropagation. In this project, I used he_uniform, which was created by Kaiming He in 2015. The he_uniform that improved Xavier initialization showed good results on my project. The batch normalization was used to the activation map after the convolution layer. The activation map was normalized to be Gaussian distribution using batch normalization. After adding four techniques, the model is shown below.

1 layer : Conv(filters 64, he_uniform) → BatchNormalization → Conv(filters 64, he_uniform) → BatchNormalization → pooling → dropout(0.25)

2 layer : Conv(filters 128, he_uniform) → BatchNormalization → Conv(filters 128, he_uniform) → BatchNormalization → pooling → dropout(0.25)

3 layer : Conv(filters 256, he_uniform) → BatchNormalization → Conv(filters 256, he_uniform) → BatchNormalization → pooling → dropout(0.25)

4 layer: GlobalMaxPooling2D → Dense(neurons 512) → dropout(0.5) → Dense(neurons 512) → dropout(0.5) → Dense(neurons 10)

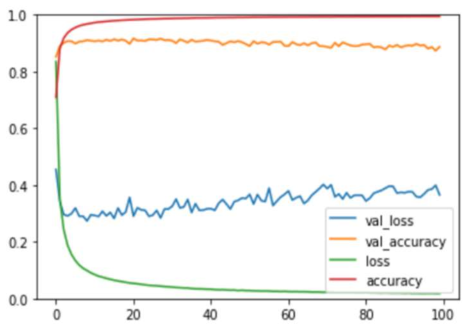

Fig. 1. The result of learning CNN.

When the model was learned, 99.5% of the accuracy and 91.6% of the validation accuracy was finally made.

To receive a variable sized image, I set the input shape to (None, None, 3) and used GlobalMaxPooling2D instead of Flatten in the fully-connected layer.

Gaussian Noise :

In this project, I made noise with Gaussian distribution to see how noise affects pre-trained models. The np.random.normal function of numpy was used to create gaussian distribution. I generated noise images from a variable named noise_param multiplied by the random numbers generated in np.random.normal, and then reflected the noise values in the image. The value of noise_param ranges from 0 to 1, but the closer it approaches 1, the more severely the original image is damaged, and the closer it is to 0, the noise is not visible well. In this project, noise_param was set to 0.1.

Fast Gradient Sign Method(FGSM) :

The FGSM used in this project belongs to the white box attack technology. The white box attack is performed under the assumption that the attacker can access all parameter values of the target model. FGSM uses the gradient of neural networks to create adversarial images. An image that maximizes the loss is generated by calculating the gradient of the loss function for the input image.

In short, the neural network model has a linear characteristic, so it is possible to sufficiently change the activation value by differentiating specific x only once in the classification model and moving as much as ε for each axis. Detailed formulas and explanations for FGSM are given in related works.

Randomization layer :

According to the paper[2] published in 2018, adding the "randomization layer" could mitigate the adversarial effect. Therefore, I try to create a randomization layer as proposed in this paper to preprocess adversarial examples.

The randomization layer consists of two sub-layers. For the first sub-layer, resize the adversarial examples to a random size when adversarial examples are entered into the resizing sub-layer. The second sub-layer is inputted with adversarial examples resized into the padding layer and randomly adds padding to the left, right, top, and bottom. To add padding, I used the ZeroPadding2D function provided by Keras. After the randomization process, the shape of the adversarial examples will have a different height and width for each image. Therefore, the modified image cannot be placed in the existing ndarray, so a new ndarray is created using np.empty.

Autoencoder :

The second way to mitigate adversarial effects is to use an autoencoder. In the second convolution layer only, the stride was designated as 2 and the rest of the convolution layer was designated as 1. After the second batch normalization, I can get a latent vector. Up sampling2d function was used to decode the latent vector. In order to finally restore the image, the number of filters are 3, filter size 1 x 1, and I used sigmoid as activation function to decode the image in the last convolution layer.

5 Results

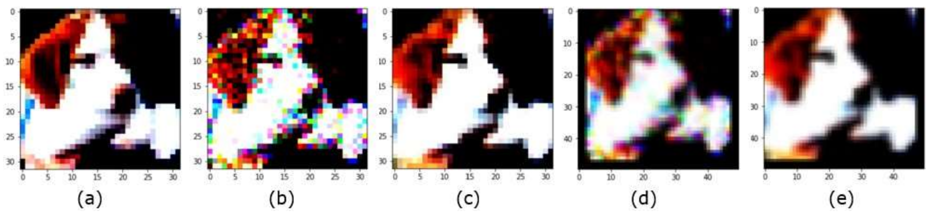

Fig. 2. Gaussian noise examples and modified adversarial examples : (a) original image; (b) gaussian noise image; (c) gaussian noise image with autoencoder applied; (d) gaussian noise image with randomization layer applied; (e) gaussian noise image with autoencoder and randomization layer applied

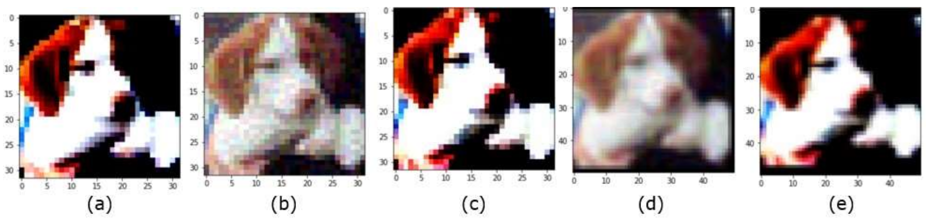

Fig. 3. FGSM examples and modified adversarial examples : (a) original image; (b) FGSM image; (c) FGSM image with autoencoder applied; (d) FGSM image with randomization layer applied; (e) FGSM image with autoencoder and randomization layer applied

(a) Original image
(b) Adversarial image
(c) Adversarial image with autoencoder applied

(d) Adversarial image with randomization method applied
(e) Adversarial image with autoencoder and randomization applied

Table 1. Gaussian Result

	a	b	c	d	e
Accuracy	91.6%	17.4%	39.6%	15.3%	21.2%

Table 2. FGSM Result

	a	b	c	d	e
Accuracy	91.7%	6%	31.7%	10.8%	20.6%

Table 1. and Table 2. can be seen accuracy results in two methods used to mitigate adversarial effects on gaussian noise and FGSM. When the randomization layer was applied, the accuracy of the image did not change much. However, as shown in c, when only autoencoder was applied, it was seen that adversarial effects were greatly mitigated.

In this project, after developing a high-performance CNN of variable length, the developed model was imported and used by transfer learning. To generate adversarial examples, Gaussian noise, which uses the most basic Gaussian distribution, was used, and adversarial examples were created by developing the most popular FGSM technique introduced by Goodfellow et al in 2015. Randomization and autoencoder were developed to mitigate the adversarial effects of adversarial examples. Two mitigating methods were put in front of the CNN in a pre-processing layer so as not to affect the performance of the CNN. Experimental results showed that randomization did not mitigate adversarial effects. In this part, there seems to be a part that I did not consider when designing the randomization layer. In contrast, the autoencoder was seen to dramatically mitigate adversarial effects. As a result, experiments have shown that using an autoencoder is a good way to mitigate adversarial effects.

References

1. T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, **"BadNets: Evaluating Backdooring Attacks on Deep Neural Networks,"** *IEEE Access*, vol. 7, pp. 47230–47243, 2019, doi: 10.1109/ACCESS.2019.2909068.
2. Xie, C., Zhang, Z., Yuille, A. L., Wang, J., & Ren, Z. (2018). **Mitigating adversarial effects through randomization.** 6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings, 1–16.
3. Goodfellow, I. J., Shlens, J., & Szegedy, C. (2015). **Explaining and harnessing adversarial examples.** 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, 1–11.