Inverted Conditional Generator Classifier

Slow but accurate and robust gradient-descent based prediction classifier

Jeongik Cho[1]

Dept. of Computer Science and Engineering[1]

College of Engineering[1]

Konkuk University, Seoul, Korea[1]

jeongik.jo.01@gmail.com[1]

Abstract

Traditional deep neural network classifier receives input data and passes through hidden layers to output predicted labels. In this paper, I propose an Inverted Conditional Generator Classifier that uses conditional generators to find a pair of condition vector and latent vector that can generate the data closest to the input data, and predict the label of the input data.

The conditional generator is a generative model that receives latent vector and condition vector, and generates data with desired conditions. A decoder of conditional VAE [1] or a generator of conditional GAN [2] can be a conditional generator.

The inverted Conditional Generator Classifier uses a trained conditional generator as it is.

The inverted conditional generator classifier repeatedly performs gradient descent by taking the latent vector for each condition as a variable and the model parameter as a constant to find the data closest to the input data. Then, among the data generated for each condition, the condition vector of the data closest to the input data becomes the predicted label.

Inverted Conditional Generator Classifier is slow to predict because prediction is based on gradient descent, but has high accuracy and is very robust against adversarial attacks [3] such as noise.

In addition, the Inverted Conditional Generator Classifier can measure the degree of out-of-class through the difference between the generated nearest data and input data. A high degree of out-of-class means that the input data is separate from the cluster of each class, or Inverted Conditional Generator Classifier has little confidence in prediction. Through this, Inverted Conditional Generator Classifier can classify the input data as out-of-class or defer classification due to the lack of confidence in prediction.

Abbreviations

Inverted Conditional Generator Classifier: ICGC

Deep Neural Network: DNN

Index Terms

Supervised learning, Artificial neural networks

Multi-layer neural network, Feedforward neural networks,

1. Introduction

Recently, the DNN classifier is used in areas where accuracy is critical, such as autonomous vehicles. However, DNN is very susceptible to adversarial attacks because it can react very sensitively to small changes [4]. In recent years, many adversarial attack methods have been studied to deceive the classifier using the instability of DNN.

In this paper, I propose a new classifier called ICGC that performs gradient descent-based prediction using a conditional generator, rather than a traditional deep neural network classifier that outputs a predicted label through a hidden layer.

The conditional generator is a generator that receives condition vector and latent vector, and generates data with the desired conditions.

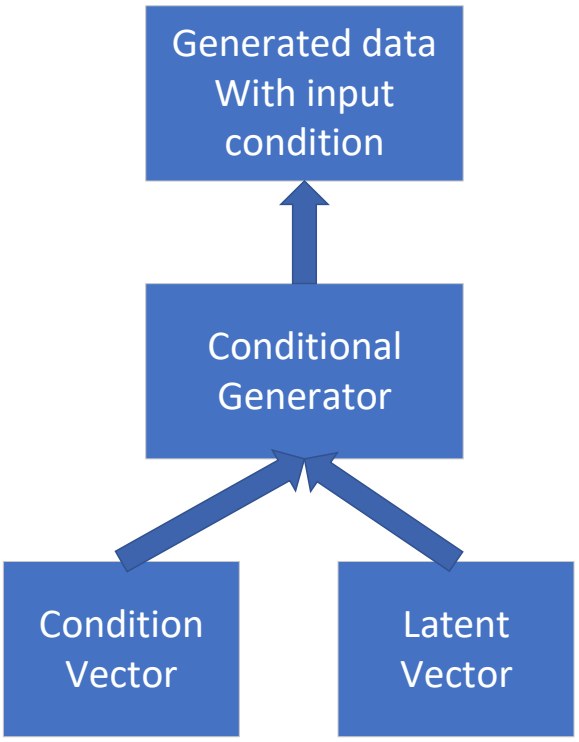

Fig.1 Conditional Generator

A decoder of conditional VAE or a generator of conditional GAN, or other conditional generative models can be a conditional generator. ICGC uses a conditional generator to find a pair of condition vectors and latent vectors that can generate the data closest to the input data through iterative gradient descent and outputs the condition vector of the data as a predicted label.

Since ICGC classifies the data by generating the data closest to the input data, it is not sensitive to small changes like the traditional DNN classifier, so it is very resistant to adversarial attacks such as noise. In particular, ICGC cannot be applied with a white-box adversarial attack that assumes a traditional DNN classifier.

The DNN classifier cannot classify the input data as out-of-class even if it belongs to out-

of-class. For example, in the case of a DNN classifier that classifies the numbers 0 to 9, when a noise image is input, it cannot be predicted as out-of-class. However, since ICGC generates the data closest to the input data among the data that the conditional generator can generate, the degree of out-of-class can be measured through the difference between the generated data and the input data.

If the input data does not belong to any class due to data modification or else, the degree of out-of-class may be high. In this case, ICGC can classify the data as out-of-class.

Or, the degree of out-of-class may be high even if the conditional generator of ICGC is not sufficiently trained or gradient descent is not sufficiently performed during prediction. In such cases, ICGC may suspend classification of the input data.

ICGC is structurally robust to adversarial attack, and can use out-of-class degree to classify modified input data as out-of-class, or withhold classification for input data with low confidence. Therefore, ICGC can be exposed to adversarial attack or can be utilized in applications where accuracy is important.

2. Inverted Conditional Generator Classifier

2.1 Training

ICGC uses trained conditional generators such as Conditional VAE or Conditional GAN as models. For conditional VAE, a decoder is used, and for conditional GAN, a generator is used as a model for ICGC. No additional training is required after training the conditional generator.

2.2 Prediction

First, ICGC finds a pair of condition vectors and latent vectors that generate data closest to input data through a latent space search. Then, among the data generated for each condition, the condition vector of the data closest to the input data becomes the predicted label.

The latent space search is to perform multiple gradient descents taking the latent vector for each condition as a variable, the model parameter as a constant, and using two losses: data difference loss and latent restriction loss. Through this, a pair of condition vectors and latent vectors that generate data close to the input data can be found.

The data difference loss is the loss to find the latent vector that can generate the data closest to the input data for each condition.

The latent restriction loss is a loss to prevent the latent vector from searching too far from the latent space used for conditional generator training.

The loss for ICGC to perform latent space search is as follows.

$$L = L_{DD} + \lambda_{LR} L_{LR}$$

$$L_{DD} = \sum_{(cnd,ltn)\in S_{in\_vec}} dif(G(cnd, ltn), in\_d)$$

$$L_{LR} = \sum_{(cnd,ltn)\in S_{in\_vec}} average\left(abs\left(z\_score(ltn)\right)\right)$$

L is the loss for ICGC to perform latent space search through gradient descent. L_{DD} is data difference loss, and L_{LR} is latent restriction loss. λ_{LR} is the weight of latent restriction loss. S_{in_vec} is a set of pairs having a cnd (condition vector) and a ltn (latent vector). S_{in_vec} has a pair of cnd corresponding to each class and ltn corresponding to the cnd as many as the number of classes. For example, if there are 10 classes, S_{in_vec} has 10 (cnd, ltn) pairs. G is a trained conditional generator. $G(cnd, ltn)$ is one data generated by G by receiving cnd and ltn. in_d is one input data. dif is a function that measures the difference between two data. z_score is a function that calculates the z score of each element of the input vector based on the distribution of latent vector used when training G. For example, when G is trained using a latent vector that follows a Gaussian distribution with mean 0 and standard deviation 1, $z\_score([1,2,-3])$ is $[1,2,-3]$. abs is a function that converts each element of the input vector to an absolute value. $average$ is a function to find the average of each element of the input vector.

To reduce L, gradient descent is performed by taking the latent vector for each condition as variables and the model parameters as constants. If gradient descent is repeatedly performed a certain number of times, the latent space search ends. Then, the difference between the data generated for each condition and the input data is measured using the dif function, and the condition with the smallest difference is determined as the predicted label.

$$(predicted\ label, latent\ vector) = \underset{(cnd,ltn)\in S_{in\_vec}}{\arg\min}\ dif(G(cnd, ltn), in\_d)$$

Label	Condition Vector				Latent vector		
num 0	1 (untrainable)	0 (untrainable)	0 (untrainable)	…	0.3 (trainable)	-1.0 (trainable)	…
num 1	0 (untrainable)	1 (untrainable)	0 (untrainable)	…	-0.2 (trainable)	0.1 (trainable)	…
num 2	0 (untrainable)	0 (untrainable)	1 (untrainable)	…	0.7 (trainable)	-0.3 (trainable)	…
…	…	…	…	…	…	…	…

Fig.2 Example of input vectors of ICGC

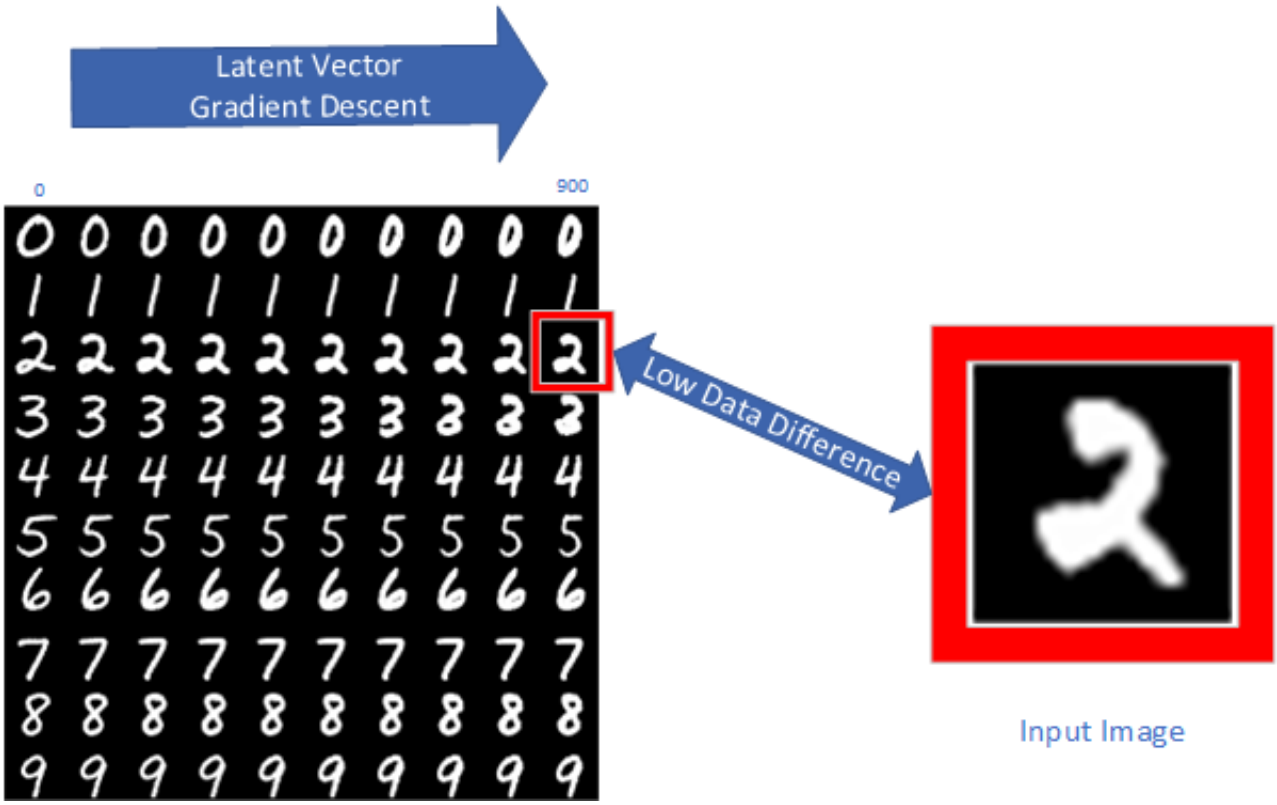

Fig.3 Prediction process of ICGC

Fig.2 is an example of an input vectors of ICGC. The condition vector, which is an untrainable variable, does not change when performing gradient descent. However, the latent vector, which is a trainable variable, changes with every gradient descent.

Fig.3 shows the process of ICGC prediction. Initially, all latent vector is initialized with the average of the latent vector distribution used during generator training. That is, at first, all latent vector for each condition are the same. Later, the latent vector changes to generate an image close to the input image. The leftmost column in Fig.3 is data generated for each condition before performing gradient descent, and the rightmost column is after gradient descent is performed 900 times. After performing a gradient descent to some extent, the input condition vector to generate data with the closest distance to the input image be the predicted label of the ICGC.

2.3 Out-of-class

DNN classifier cannot distinguish data that does not belong to any class. For example, in the case of a classifier that classifies the numbers 0 to 9, the classifier will predict the class as one of the numbers 0 to 9 even if noise is input instead of numbers.

However, since ICGC generates the data closest to the input data among the data that the conditional generator can generate, the degree of out-of-class can be measured through the difference between the generated data and the input data.

$$ooc = \min_{(cnd,ltn)\in S_{in\_vec}} dif(G(cnd, ltn), in\_d)$$

ooc is the degree of out-of-class. If the conditional generator of ICGC is sufficiently trained and sufficient gradient descent is achieved during prediction, ICGC can classify data with large ooc as out-of-class. If ICGC performance is considered insufficient, ICGC may withhold classification for data with large ooc.

2.4 Multi-label classification

In multi-class classification with one label, ICGC can predict the label of one data by creating pairs of condition vector and latent vector as many as the number of classes of the label. However, in the case of multi-label classification, the time required for prediction may be too long because there are so many possible combinations of condition vectors.

Instead, the ICGC can shorten the time for prediction by repeating prediction for each label. That is, when performing prediction on one label, the condition vector for the label to be predicted is set as an untrainable variable, and the condition vectors for the remaining labels and latent vector are set as trainable variables to perform latent space search. This prediction must be repeated as many as the number of labels.

2.5 Parallel ICGC

Gradient descent-based search always has the potential to converge to local optima, not global optima. Likewise, there is a possibility that during the latent space search by ICGC, the latent vector falls into the local optima, not the global optima.

To increase the probability that the ICGC finds a latent vector falling into the global optima, or even a little better local optima, Parallel ICGC can be used. ICGC searched one latent vector per condition, but Parallel ICGC searched multiple latent vectors per condition to perform a latent space search. In addition, a latent vector corresponding to each condition of ICGC is initialized with the average of latent vectors used in conditional generator training, but parallel ICGC latent vectors are randomly initialized with the latent vector of ICGC to find different local optima.

Label	Condition Vector				Latent Vector		
num 0	1 (untrainable)	0 (untrainable)	0 (untrainable)	...	0.0 (average) (trainable)	0.0 (average) (trainable)	...
num 0	1 (untrainable)	0 (untrainable)	0 (untrainable)	...	0.7 (random) (trainable)	-0.2 (random) (trainable)	...
num 0	1 (untrainable)	0 (untrainable)	0 (untrainable)	...	-0.6 (random) (trainable)	0.1 (random) (trainable)	...

num 1	0 (untrainable)	1 (untrainable)	0 (untrainable)	…	0.0 (average) (trainable)	0.0 (average) (trainable)	…
num 1	0 (untrainable)	1 (untrainable)	0 (untrainable)	…	0.7 (random) (trainable)	-0.8 (random) (trainable)	…
…	…	…	…	…	…	…	…

Fig.4 Example of initialized input vectors of Parallel ICGC

3. Experiment

In this experiment, the MNIST handwriting number dataset [5] was used (60000 images for training, 10000 images for test, 28x28x1 resolution). Tensorflow 2.1 without compile option and rtx2080ti was used for the experiment.

3.1 Training

3.1.1 ICGC Training

I used conditional activation GAN [6] with LSGAN [7] adversarial loss to train conditional generator. The generator receives a 10-dimensional condition vector and a 256-dimensional latent vector. All elements of the latent vector used in training follow the Gaussian distribution with mean = 0 and standard deviation = 1. The average FID [8] for each condition of the generator after training was measured to be 2.0. Since the MNIST dataset has one channel and their resolution is too low for the inception network, the width, height, and channel are tripled for the FID evaluation (84 × 84 × 3).

3.1.2 DNN classifier Training

The DNN classifier is a general classifier that consists of a convolution layer for all hidden layers except the output layer and uses cross-entropy as a loss function. The trained DNN classifier showed 99.43% accuracy for the MNIST test dataset.

3.2 Evaluation

I evaluated the accuracy of ICGC and DNN classifier according to the intensity of gaussian noise and FGSM [10] noise. The original image is normalized from -0.5 to 0.5. The Gaussian noised image is as follows.

$$gaussian\ noised\ image = clip\big(original\ image + \sigma * clip(gaussian\ noise)\big)$$

$gaussian\ noise$ is gaussian noise with an average of 0 and a standard deviation of 1, and a $clip$ is a function that clips the input to maintain a range of -0.5 to 0.5. σ represents the intensity of noise.

The FGSM noised image of the DNN classifier is as follows.

$$FGSM\ loss = cross\ entropy(C(in\_d), label)$$

$$FGSM\ noise = sgn\left(\frac{\Delta FGSM\ loss}{\Delta in\_d}\right) \div 2$$

$$FGSM\ noised\ image = clip(original\ image + \sigma * FGSM\ noise)$$

in_d is input data. C is a classifier. The $label$ is the label of in_d. sgn is sign function.

In the case of ICGC, noise that fits the definition of FGSM cannot be implemented, so FGSM loss for ICGC predict was used.

$$FGSM\ loss = -L_{DD} - \lambda_{LR}L_{LR} + 2 \times \left(dif(G(label, ltn), in\_d) + \lambda_{LR} average\left(abs(z\_score(ltn))\right)\right)$$

$$FGSM\ noise = sgn\left(\frac{\Delta FGSM\ loss}{\Delta in\_d}\right) \div 2$$

$$FGSM\ noised\ image = clip(original\ image + \sigma * FGSM\ noise)$$

ltn is a latent vector corresponding to the label after ICGC performs prediction. ICGC's FGSM noise increases ICGC's loss on real labels and lowers ICGC's loss on other labels.

For prediction of ICGC, gradient descent was performed 100 times for each image, and Adam optimizer [9] (learning rate = 0.001, beta1=0.9, beta2 = 0.999) was used. The latent restriction loss (λ_{LR}) is 0.03 and the dif function is mean absolute error.

Since the prediction of ICGC is slow, for fast experiment, ICGC used only 1000 randomly selected data from the MNIST test dataset for evaluation. The prediction of the DNN classifier was fast enough, so all 10000 MNIST test data were used for evaluation.

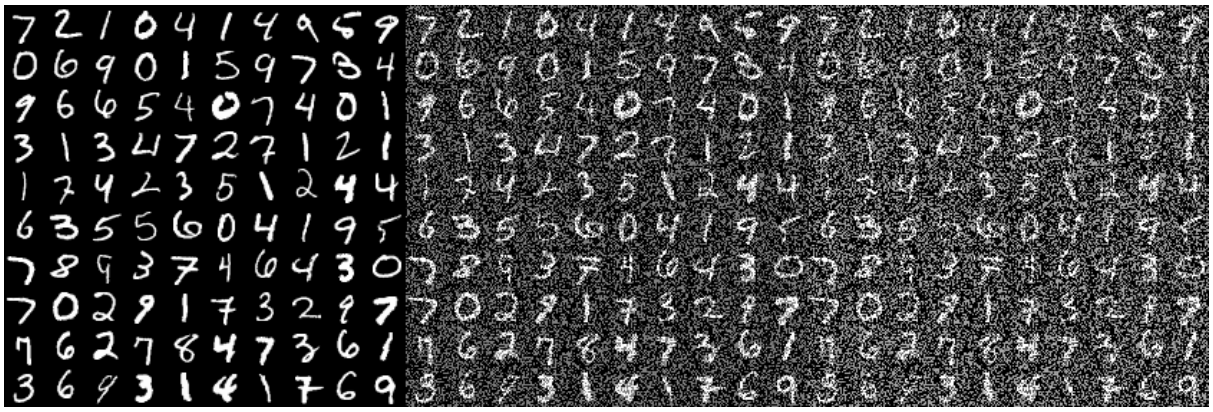

Fig.5 Gaussian noised images examples. $\sigma = 0.0$, $\sigma = 1.0$, $\sigma = 1.3$ in turn

3.2.1 Parallel ICGC test

I compared the accuracy of parallel ICGC and non-parallel ICGC for Gaussian noised images.

Gaussian noise								
Sigma	0	0	0.2	0.2	0.4	0.4	1	1
ICGC Latent vector / condition vector	1	10	1	10	1	10	1	10
ICGC accuracy (%)	94.2	95.9						91.5
ICGC average ooc	0.028786	0.025318						0.212831
ICGC time (sec)	2627	6668						6612

Fig.6 Gaussian noised image test

Fig. 6 shows that parallel ICGC has better performance than normal ICGC.

Next, I compared the accuracy of parallel ICGC and DNN classifiers for gaussian noise and FGSM noise.

3.2.2 ICGC test

Gaussian noise							
Sigma	0	0.2	0.4	0.6	0.8	1	1.2
ICGC Latent vector / condition vector	10					10	10
ICGC accuracy (%)	95.9					91.5	87.9
DNN classifier accuracy (%)	99.43	99.37	99.27	99.09	97.61	93.63	84.39
ICGC average ooc	0.025318					0.212831	0.250656
ICGC time (sec)	6668					6612	6598
DNN classifier time (sec)	17	17	17	17	17	17	17

Fig.7 Gaussian noise compare

FGSM noise					
Sigma	0	0.1	0.2	0.3	0.4
ICGC Latent vector / condition vector	10	10	10	10	10
ICGC accuracy (%)	95.9		95.3		92.4
DNN classifier accuracy (%)	99.43	96.15	82.28	54.12	27.49
ICGC average ooc	0.025318		0.109149		0.195751
ICGC time (sec)	6668		13308		12911
DNN classifier time (sec)	17	26	26	26	27

Fig.8 FGSM noise compare

FGSM swap noise				
Sigma	0		0.4	0.6
ICGC Latent vector / condition vector	10		10	10
ICGC accuracy (%)	95.9		90.2	79.5
DNN classifier accuracy (%)	99.43		98.7	95.4
ICGC average ooc	0.025318		0.108271	0.149172
Time (sec)	6668		13751	13659.39

Fig.9 FGSM swap noise

Fig.7 and Fig.8 show the accuracy of parallel ICGC and DNN classifier for Gaussian noise and FGSM noise. Fig. 9 shows the accuracy measured by swapping the FGSM noised images of ICGC and DNN classifier for the same data. In Fig.7, Fig.8, and Fig.9, the accuracy of DNN classifier is higher when sigma is low, but ICGC accuracy is higher when sigma is over a certain value.

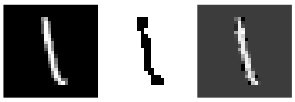

Fig.10 Original image, ICGC FGSM noise, noised image in turn. $\sigma = 0.4$

Fig.11 Original image, DNN classifier FGSM noise, noised image in turn. $\sigma = 0.4$

Fig. 10 and Fig. 11 show examples of FGSM noise and noised images of ICGC and DNN classifiers. The FGSM noise of the DNN classifier is noise that humans cannot understand, but the FGSM noise of ICGC is similar to the inverted image of the original image.

Fig.12 ICGC prediction

Fig. 12 shows the images generated by ICGC to predict the gaussian noised image on the right.

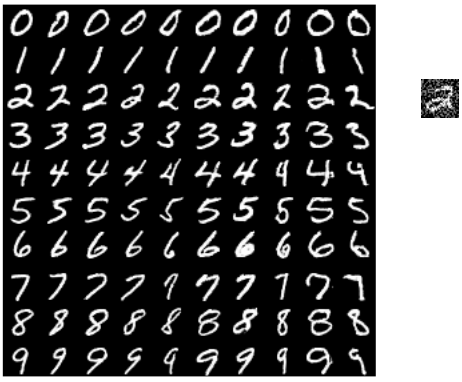

Fig.13 Parallel ICGC prediction

Fig. 13 shows images generated by parallel ICGC with $\frac{latent\ vector}{condition\ vector} = 10$ to predict the label of the gaussian noised image on the right.

3.2.3 ooc test

Gaussian ooc								
Threshold	INF	0.06	INF	0.06	INF	0.06	INF	0.06
ICGC Latent vector / condition vector	10	10	10	10	10	10	10	10
Sigma	0	0	0.1	0.1	0.2	0.2	0.4	0.4
Correct	959	944		917		567		0
Wrong	41	39		18		5		0
ooc	0	17		65		428		1000
correct / (wrong + correct) * 100	95.9	96.03		98.07		99.13		NAN
ooc / (wrong + correct + ooc) * 100	0	1.7	0	6.5	0	42.8	0	NAN

Fig.14 ooc test

Fig. 14 shows the accuracy when ICGC classified data with ooc over the threshold as ooc. When holding or discarding data with high ooc, accuracy $\left(\frac{correct}{(wrong+correct)} * 100\right)$ is higher than without ooc.

4. Conclusion

ICGC is slow when predicting because it predicts based on gradient descent, but accuracy is high and very robust against adversarial attacks. In particular, ICGC cannot be applied with a white-box adversarial attack that assumes a traditional DNN classifier. Also, unlike DNN, by using the degree of out-of-class, ICGC can classify the modified data as out-of-class or withhold classification for data with low confidence.

5. References

[1] Kihyuk Sohn, Honglak Lee, Xinchen Yan

Learning Structured Output Representation using Deep Conditional Generative Models

https://papers.nips.cc/paper/5775-learning-structured-output-representation-using-deep-conditional-generative-models

[2] Mehdi Mirza, Simon Osindero

"Conditional Generative Adversarial Nets", arXiv preprint arXiv:1411.1784, 2014.

https://arxiv.org/abs/1411.1784 (accessed 16 February 2020)

[3] Xiaoyong Yuan, Pan He, Qile Zhu, Xiaolin Li

Adversarial Examples: Attacks and Defenses for Deep Learning

https://arxiv.org/abs/1712.07107

[4] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, Rob Fergus

Intriguing properties of neural networks https://arxiv.org/abs/1312.6199

[dataset] [5] Yann LeCun, Corinna Cortes, Christopher J.C. Burges

THE MNIST DATABASE of handwritten digits

http://yann.lecun.com/exdb/mnist/

[6] Jeonglk Cho, Kyoungro Yoon

Conditional Activation GAN: Improved Auxiliary Classifier GAN

http://vixra.org/abs/1912.0204

[7] Xudong Mao, Qing Li, Haoran Xie, Raymond Y.K. Lau, Zhen Wang, Stephen Paul Smolley

Least Squares Generative Adversarial Networks

The IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2794-2802

https://ieeexplore.ieee.org/document/8237566

[8] Heusel, Martin and Ramsauer, Hubert and Unterthiner, Thomas and Nessler, Bernhard and Hochreiter, Sepp

GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium Advances in Neural Information Processing Systems 30 (NIPS), 2017, pp. 6626-6637

https://papers.nips.cc/paper/7240-gans-trained-by-a-two-time-scale-update-rule-converge-to-a-local-nash-equilibrium

[9] Diederik P. Kingma, Jimmy Ba

Adam: A Method for Stochastic Optimization

https://arxiv.org/abs/1412.6980

[10] Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy

Explaining and Harnessing Adversarial Examples

https://arxiv.org/abs/1412.6572