Enhancing Adversarial Attack Methods Against Machine Learning Models Using Similarity Index Measures

Erfan Yazdpour
Student ID: 20924364
erfan.yazdpour@uwaterloo.ca

Mahsa Javadi
Student ID: 20919498
mjavadi@uwaterloo.ca

Katayoun Hossein Abadi
Student ID: 20915759
k6hosseinabadi@uwaterloo.ca

ABSTRACT

Adversarial examples are inputs that are purposefully constructed to cause a neural network to misclassify them. The challenge in generating these inputs using non-gradient-based methods is the lack of efficiency in terms of execution time and performance. In this paper, we propose an improvement using Structural Similarity Index Measure (SSIM) and Information theoretic-based Statistic Similarity Measure (ISSM) on different non-gradient-based methods including methods based on fuzzing and genetics algorithms. After implementing and doing experiments for comparing the effect of these measures, it was found out that adding Gumbel statistical distribution and the use of similarity measures can have a huge positive impact on the execution time of fuzzing and evolutionary based methods on some machine learning models. All the related codes to implementations and experiments can be found at `https://github.com/AAGen.`

1. INTRODUCTION

Deep Learning is becoming a prevalent tool for training the data because of its prominent performance. However, as deep models are vulnerable to attacks, adversaries can cause the model to make mistakes by attacking the deep models. [1]Therefore, it is of significant importance to work on methods to determine and find these adversarial attacks in order to defend against them.

Machine learning models can be led to wrong prediction or misclassification just by adding a small perturbation to the input of the model, imperceptible for humans, but sensitive enough for the model to change its behavior. This altered input is called adversarial input and usually, it is designed to make a machine learning model misbehave or misclassify. An adversarial attack introduces maliciously designed data to deceive an already trained model.

These adversarial inputs created a new category of attacks and caused a new set of security problems that can even be life-threatening in many cases such as self-driving cars' machine learning models. As an example, Figure 1. An example of adversarial attack on a neural network indicates that a classification model can be tricked to classify an input to a wrong category just by adding a small effective perturbation which is not recognizable by human eyes but can completely mislead the machine learning classification model so that it will classify what was at first a panda as a gibbon.

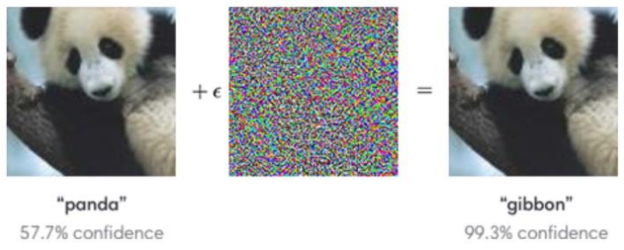

Figure 1. An example of adversarial attack on a neural network [2]

To this purpose, test, analysis and verification (TAV) procedures have been developed which are adjusted to attack and make the deep models and neural networks more resistant. In this project, we will propose a method to improve the performance of some non-gradient-based method for generating adversarial examples by using Structural Similarity Index Measure (SSIM) and Information theoretic-based Statistic Similarity Measure (ISSM). We also will study the effect of these measures on different non-gradient-based methods to find adversarial examples to different neural networks. These non-gradient-based methods are based on fuzzing and genetics algorithms.

2. Contributions

In this project report, we make the following contributions:

1. A brief survey to generate adversarial examples: In this report, we briefly study different methods for finding adversarial examples focusing on non-gradient-based methods such as fuzzing-based and genetics/evolutionary methods. This is done in sections 3 and 4
2. Proposing new method and measures to enhance the performance of non-gradient-based methods: In this report, Structural Similarity Index Measure (SSIM) and Information theoretic-based Statistic Similarity Measure (ISSM) are used to enhance different methods that are based on fuzzing and genetics/evolutionary algorithms. The description of these new approaches can be found in section 4.
3. An extensive analysis on effects of two new proposed measures on different TAV methods that are non-gradient-based: We implement all the considered methods for finding adversarial examples for different models and datasets to evaluate the effectiveness of this new approach. The information about the evaluation can be found in section 5 and 6.

3. Background and Related Works

The reason of vulnerability of neural networks to adversarial attacks was studied by Goodfellow et al. [3] who argued that the reason is not because of overfitting and non-linearity of the model but the reason of that is the high dimensionality of the input space and local linearity of the model.

Adversarial attacks were first studied by Szegedy et al. [4] who performed adversarial attacks on images by adding an imperceptible noise to the input and misguided deep learning models successfully. After that many methods are proposed to find these attacks to make the model robust to them. They are gradient-based and non-gradient-based techniques in this area. Gradient-based techniques such as the Fast-Gradient Sign Method (FGSM) [5], the Basic Iterative Method (BIM) proposed by Kurakin et al. [6], DeepFool proposed by Moosavi Dezfooli et al. [7], and the Jacobian-based Saliency Map Attack (JSMA) [8] rely on the detailed model information. However, non-gradient-based techniques such as fuzzing methods [9] [10], evolutionary processes and approaches [11], logic-solvers [12], local search [13] and genetic algorithm methods do not use gradient information to find adversarial examples, and these attack algorithms that do not need to follow the gradient have an advantage in such areas. So, non-gradient-based techniques are more effective against defenses that seek to obscure the gradient to defeat gradient-based attacks. Attack algorithms that do not need to follow the gradient have an advantage in such areas.

Adversarial inputs are always existing inside the input region of neural networks and machine learning models and that is why developers and machine learning model holders are trying so hard to test their models to find these adversarial inputs so that they can defend against these attacks.

Adversarial attacks can be classified into Targeted and Untargeted attacks according to the adversarial goals. Targeted attacks are the attacks that tries to mislead the model to predict a particular class other than the true class. On the other hand, Untargeted attacks are the attacks that tries to mislead the model to predict any of the incorrect classes [14]. Also, adversarial attacks can be classified into White-box attacks and black-boxed attacks based on adversarial knowledge. White-box attacks are attacks that knows the architecture of the model, parameters of the model, training dataset etc. However, Black-box attacks are attacks that does not have access to the architecture and parameters of the model, training dataset etc. [14].

In this project, we focused on non-gradient-based techniques focusing on fuzzing methods and genetic algorithm methods and trying to improve the performance of these methods by adding similarity index techniques. Therefore, our studies and works main difference is on adding similarity index measures such as structural similarity index measure (SSIM) and information theoretic-based Statistic Similarity Measure (ISSM) to different fuzzing and genetic algorithm methods used for finding adversarial examples. In addition, we evaluate our techniques on four different models which includes fully connected and convolutional neural networks on both MNIST and CIFAR10 datasets to determine the success of our proposed enhancements.

4. Methods

A. *Similarity index measures*

Image similarity is fundamental to a wide range of applications in image processing and computer vision. The purpose of image similarity is to produce methods for objective assessment of quality versus subjective human image-quality evaluation [15]. Image similarity compares two images to detect how visually they are similar [16]. There are various methods for the purpose of image similarity such as Root mean square error (RMSE), Structural similarity index measure (SSIM), Signal to reconstruction error ratio (SRE), Information theoretic-based Statistic Similarity Measure (ISSM), Spectral angle mapper (SAM), Universal image quality index (UIQ), Feature-Based Similarity Index (FSIM), Peak signal-to-noise ratio (PSNR). In this project we used structural similarity index measure (SSIM) and Information theoretic-based Statistic Similarity Measure (ISSM).

1) Structural similarity index measure (SSIM)

The structural similarity index measure is introduced by Wang, Bovik, Sheikh, & Simoncelli, 2004 [17]. It is based on the statistical measurements. It is used the mean and the standard deviation to extract the statistical image features for image similarity purpose. The formula for SSIM to get the similarity between two images is as follows [15]:

$$S(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_1)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Where $S(x,y)$ is the structural similarity index measure between the test image(x) and the training image (y). μ_x and σ_x^2 are the mean and the variance of the pixels in the test image x, respectively. μ_y andσ_y^2 are the mean and the variance of the pixels in the training image y, respectively. c_1 and c_2 are constant values.

2) Information theoretic-based Statistic Similarity Measure (ISSM)

The Information theoretic-based Statistic Similarity Measure is introduced in 2014 [15]. ISSM is an image similarity measure which is based on the combined of the statistical and information theory approach for image similarity while SSIM is only based on statistical direction. So, the main properties of ISSM are the extraction of the statistical properties of the image by the statistical method (SSIM) and theoretical properties of the image using the theoretical approach (Shannon entropy and joint histogram).

B. Gumbel Distribution

A Gumbel distribution is used to model the largest value from a relatively large set of independent elements from distributions whose tails decay relatively fast, such as a normal or exponential distribution [18].

Gumbel Distribution represents the distribution of extreme values either maximum or minimum of samples used in various distributions. It is used to model distribution of peak levels. It is a popular, asymmetric, extreme value distribution (EVD), used to model maximums and minimums [19] [20].

The pdf of the Gumbel distribution with location parameter μ and a scale parameter β is as follows:

$$f(x) = \frac{1}{\beta} exp\left(-\left(\frac{x-\mu}{\beta} + exp\left(-\frac{x-\mu}{\beta}\right)\right)\right)$$

Equation 1. Gumbel Distribution

Figure 2 shows a graph of the Gumbel distribution for different values of μ and β.

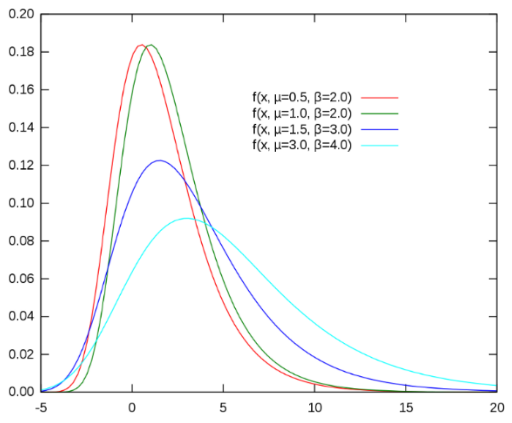

Figure 2. Gumbel Distribution [21]

1) Characteristics of the Gumbel Distribution

Some of the specific characteristics of the Gumbel distribution are as follows: [22] [23] [24]

- As we can see in Figure 2, the shape of the Gumbel distribution is skewed to the left.
- As mentioned earlier, the Gumbel pdf has location parameter μ which is equal to the mode but differs from median and mean. This is because the Gumbel distribution is not symmetrical about its μ.
- From the Equation 1, it can be seen that the pdf of Gumbel distribution has no shape parameter. So, it has only one unchanging shape which shifts according to the location parameter μ.
- It shows in Figure 2 that as μ increases, the distribution shifts to the left; As μ decreases, it shifts to the right.
- In Figure 2 we can see that as σ increases, the pdf spreads out and becomes shallower. As σ decreases, the pdf becomes taller and narrower.

C. Fuzzing-Based Methods

One of the popular methods used for creating adversarial examples is fuzzing method [25], so we developed some existing fuzzers as well as proposed a new fuzzer. All the fuzzers that we developed are based on randomness. One of them is completely random, three of them are built with different statistical distribution such as Normal, Laplace and Gumbel, and the other fuzzers that we proposed are combination of randomness and similarity index measures.

We used similarity index measures [15] for developing these fuzzers. Similarity index measure is used for image processing and has different indexes, we implemented it on the fuzzers and after lots of trial and errors, between similarity indexes we used two indexes which are SSIM and ISSM, because they were effective for fuzzers' operation, and help fuzzers create adversarial examples in less iteration and time based on our experiments.

The list of the implemented fuzzers is as follows:

1	Completely random fuzzer
2	Laplace random fuzzer
3	Normal random fuzzer
4	Gumbel random fuzzer
5	Completely random fuzzer + SSIM
6	Laplace random fuzzer + SSIM
7	Normal random fuzzer + SSIM
8	Gumbel random fuzzer + SSIM
9	Completely random fuzzer + ISSM
10	Laplace random fuzzer + ISSM
11	Normal random fuzzer + ISSM
12	Gumbel random fuzzer + ISSM

Table 1. List of implemented fuzzing-based methods

1) completely random fuzzer

As mentioned earlier, one of the methods we used for generating adversarial examples is completely random fuzzer. In this method, we defined an epsilon variable which is the changing amount, also we created a vector which is a list of random numbers, including -1, 0, 1 by using "randint" function in python programming language, and these random numbers changes the direction. So, in this fuzzer, pixels are changed with the amount of epsilon in the random direction. For changing the picture, we added noise to the main image in the direction of these random numbers with the amount of epsilon and created a new image. This part is implemented in python by 'Image += epsilon * vec line of code'.

A new image was given to the model as an input, and the result was compared to the main image. If the prediction of the model was not the same as the main image, we figured out that an adversarial example is founded.

2) Random Fuzzer with Laplace Statistical Random Distribution

A Laplace distribution is continuous probability distribution for a real-valued random variable. The probability density function (pdf) of a Laplace distribution is defined as follows:

$$f(x) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$$

Equation 2. Laplace Distribution

Where μ is a location parameter (the mean) and $b > 0$ is a scale parameter (the diversity). In Figure 3. A Laplace distribution , Laplace distributions with different location and scale parameters is shown. As it can be seen, the location parameter moves the distribution to the left or right, and the scale parameter changes the peak of the distribution.

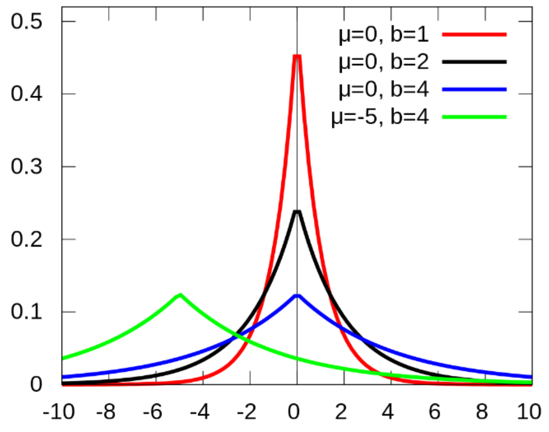

Figure 3. A Laplace distribution [26]

In this method, we added noise with the help of Laplace statistical distribution. We created a vector with Laplace distribution with a mean of 0 and variance of epsilon which is the range that we can use for changing the pixels in random basis. So, we just defined the amount of epsilon but changes in amount and direction are random variables. This part is implemented in python by:

'vec = np.random.laplace (0, epsilon, imageFlattenedShape)'

This vector is added to the main image to create a new image, and then the new image is given to the model as an input, and the result is compared to the main image, if they are different, the adversarial example is found.

The difference between this fuzzer and completely random fuzzer is that in completely random fuzzer, there is a random direction with the amount of epsilon, but in this method Laplace distribution chooses the direction and the number of changes.

The overall operation of the remaining fuzzers such as Normal and Gumbel are same as the Laplace fuzzer, but the statistical distribution is different. The Gumbel distribution was defined in the previous section of the report. Below is the brief description of a normal distribution.

A Normal distribution also called a Gaussian distribution is continuous probability distribution for a real-valued random variable. The probability density function of a normal distribution is defined as follows:

$$f(x) = \mathcal{N}(\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$$

Equation 3. Normal Distribution

Where μ is the mean and σ^2 is the variance of the distribution (σ is the standard deviation). As it can be seen in Figure 4, the mean controls the location of the peak of the distribution and the variance controls the width of the distribution. [25]

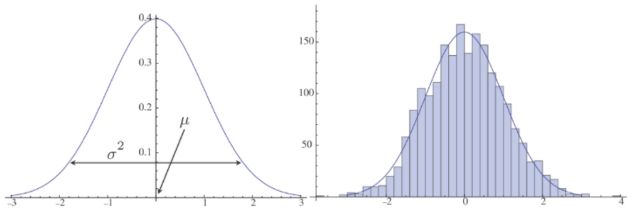

Figure 4. (Left) A normal distribution with mean 0 and variance 1. (Right) A bell curve approximating a data set (not normalized) [27]

The comparison of Normal and Laplace distribution is shown in Figure 5.

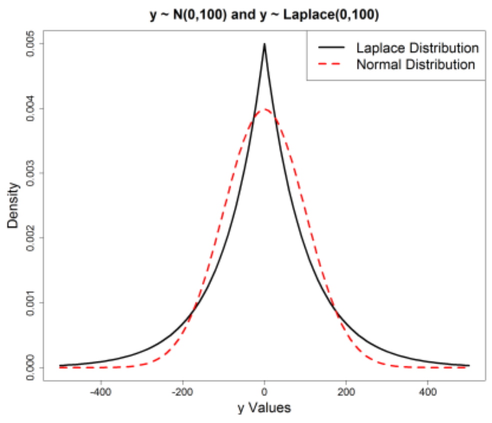

Figure 5. Comparison of Normal and Laplace distribution [28]

3) Developing Fuzzers Using Similarity Index Measures

For developing the fuzzers, we used similarity indexes such as SSIM and ISSM to measure the similarities between images.

The goal is to decrease the similarity between a new image and the main image to help the model classify the image wrongly.

The process is as below:

First, we assumed that the image is 100% the same as itself (similarity 1 = 1). Then, we added noise to the picture and gives it as an input to the models to create a new image. The process of adding noise to the images is the same as the previous fuzzers. At the next step, we measured the similarity of the new image with the main image by SSIM and ISSM indexes and named it similarity 2. If the similarity is decreased (similarity2<similarity1), it is a good result, and we keep the changed image and continue the process until we create an image that make the model predict wrongly. So, we use similarity 1 = similarity 2 and again we repeat the previous process until the image changes in a way that the model classifies it wrongly, then an adversarial example is generated. But if the similarity is increased that noise is not appropriate, and we try to generate another noise.

All in all, we concluded that the fuzzers with the similarity index measure have much better results than the ones without the similarity index measure.

D. Evolutionary/Genetics Algorithm Based Method

Based on previous works on methods using genetics algorithm, we tried to implement both basic methods and our new proposed methods so that we can compare them all together. It should be mentioned that all these methods are in the category of targeted adversarial attacks comparing to fuzzing based methods which were categorized under untargeted adversarial attacks. The basic method consists of 5 different steps:

1. Initiating the primitive population using a random factor (in this case we have used a completely random vector to alter the original image)
2. Crossing over two parents to generate two new children and adding them to the pool.
3. Mutation with a certain rate
4. Evaluating items in the pool with a fitness function which is comparing each item to the original image using similarity measures.
5. Selecting top 30% of the pool based on the evaluation process on the last step.

This process may get repeated for a certain limited cycles (we set ours to 5000 iterations) and we might get the result with the minimum qualification that we need which is for our ML Engine to misclassify our image and detect it as our target with a certain accuracy (we set this to more than 30%) earlier than maximum number of iterations. It should be noted that our goal is to make all the population adversarial and then pick the best one from them, so our methods won't stop unless they reach to the iteration limit or all of our items in the pool are adversarial examples.

Then we tried to use our similarity measure (SSIM) for the crossover part in a way that based on this metric all new children must be less similar to the original image than their parents.

We also tried to use the Gumbel statistical distribution for generating our initial population to gain an advantage there in terms of having more meaningful initial population comparing to generating them completely randomly.

At the end the described development phase above resulted in 4 different evolutionary/genetics algorithms-based methods for finding adversarial examples:

1	Basic Genetic Adversarial
2	Basic Genetic Adversarial + SSIM
3	Basic Genetic Adversarial + Gumbel
4	Basic Genetic Adversarial + Gumbel + SSIM

Table 2. List of implemented evolutionary methods

6. Evaluation

Our evaluations implemented using python 3 and the Keras framework [29] for Tensorflow on GPU. The GPU configuration is NVIDIA-SMI 470.57.02.

A. *Model and Benchmarks*

In this project, we used a fully connected and a convolutional neural network architecture on MNIST and CIFAR10 datasets. So, we trained four models to examine the efficiency of our approach on different models. To train these models we used keras library in python programming language. Keras is an API for developing artificial neural networks that runs on top of the Tensorflow engine [29] [30].

One of the main issues in deep learning is overfitting, which happens when the accuracy of the training data is much higher than the testing accuracy [31] [32]. To avoid overfitting, we can do regularization, data augmentation, and tuning the hyperparameters [31] [32]. By regularization, the generalization error is decreased which helps preventing overfitting. We did regularization by adding dropout layers in our model, which includes dropping some neurons in every epoch. We chose to drop randomly 20% and 30% of data in each epoch. In data augmentation, we use existing data to generate alternate ones and use them in the training process in order to give the neural network a wide variety of inputs, which helps in avoiding overfitting [33]. We did data augmentation by generating images using "ImageDataGenerator" function in keras library. In hyperparameter tuning, we should choose the activation function, batch size, percentage of dropouts, number of neurons and epochs, and the optimizer to gain the best result and accuracy. The activation function normalizes the data and add nonlinearity to the input data. The most popular activation function is "relu" which we used in training the models. Batch size is the number of samples used in training process which is chosen 32 in this project. Dropout percentage is the percent of data that dropped randomly in each epoch, and we chose 0.2 and 0.3 for training models. In order to update the parameters of the model such as weights and biases, we use "Adam" optimizer. For convolutional neural networks, we used 3*3 filter size, and 2*2 max pooling.

The plots of the accuracy and loss of each model are as follows:

- Fully connected neural network (FCNN) on MNIST dataset:

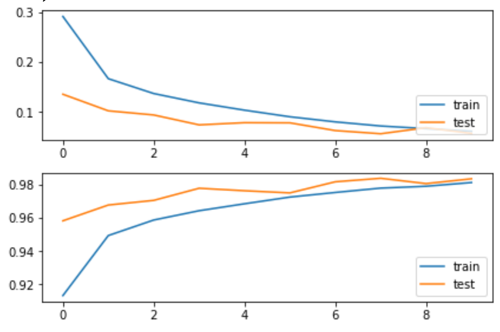

Figure 6. Loss and accuracy of FCNN on MNIST

- Convolutional neural network (CNN) on MNIST dataset:

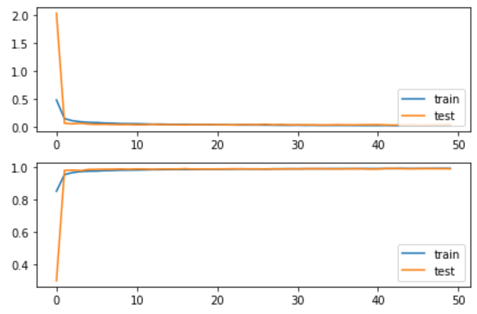

Figure 7. Loss and accuracy of CNN on MNIST

- Fully connected neural network (FCNN) on CIFAR10 dataset:

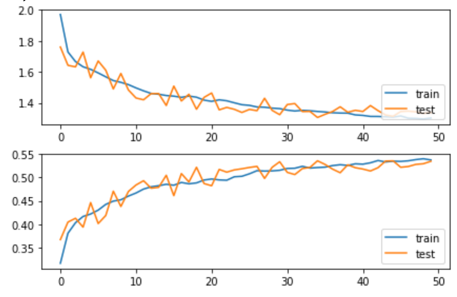

Figure 8. Loss and accuracy of FCNN on CIFAR10

- Convolutional neural network (CNN) on CIFAR10 dataset:

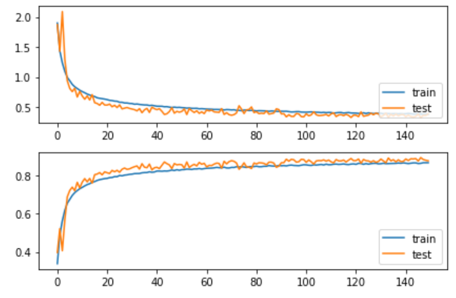

Figure 9. Loss and accuracy of CNN on CIFAR10

As the results showed, the convolutional neural networks are performed much better than fully connected neural networks. For MNIST dataset, the accuracy of the model for FCNN is 98% while for CNN is 100%. For CIFAR10 dataset, the accuracy of the model for FCNN is 55% while for CNN is over 80%. Also, fully connected neural networks are more likely to overfit and it takes much more time to train than convolutional neural networks because there are more parameters to train.

B. Results

We focused on the time execution while maintaining the image quality as the same as the output of other existing methods and hence the following is our results:

Neural Network Type: FCNN		**Dataset:** MNIST
Used Similarity Measure: None		
Method	**Iteration Number**	**Execution Time**
Gumbel	25	1.0632 secs
Laplace	165	4.9727 secs
Normal	298	9.0562 secs
Random	420	12.9225 secs

The best image quality of output image was for "Gumbel" comparing to other methods.

Neural Network Type: CNN		**Dataset:** MNIST
Used Similarity Measure: None		
Method	**Iteration Number**	**Execution Time**
Gumbel	52	1.6420 secs
Laplace	499	15.5769 secs
Normal	906	28.1524 secs
Random	1693	52.3344 secs

The best image quality of output image was for "Gumbel" comparing to other methods.

Neural Network Type: FCNN		**Dataset:** MNIST
Used Similarity Measure: SSIM		
Method	**Iteration Number**	**Execution Time**
Gumbel	26	2.3437 secs
Laplace	141	5.0605 secs
Normal	267	9.8361 secs
Random	311	11.6539 secs

The best image quality of output image was for "Gumbel" comparing to other methods.

Neural Network Type: CNN		**Dataset:** MNIST
Used Similarity Measure: SSIM		
Method	**Iteration Number**	**Execution Time**
Gumbel	53	2.3437 secs
Laplace	416	16.0198 secs
Normal	746	29.8130 secs
Random	773	31.5288 secs

The best image quality of output image was for "Gumbel" comparing to other methods.

Neural Network Type: FCNN		**Dataset:** MNIST
Used Similarity Measure: ISSM		
Method	**Iteration Number**	**Execution Time**
Gumbel	28	1.1254 secs
Laplace	201	7.2591 secs
Normal	358	13.0692 secs
Random	519	18.9422 secs

The best image quality of output image was for "Gumbel" comparing to other methods.

Neural Network Type: CNN		**Dataset:** MNIST
Used Similarity Measure: ISSM		
Method	**Iteration Number**	**Execution Time**
Gumbel	51	1.9451 secs
Laplace	402	14.8262 secs
Normal	971	35.4905 secs
Random	1794	65.6065 secs

The best image quality of output image was for "Gumbel" comparing to other methods.

Neural Network Type: FCNN		**Dataset:** CIFAR10
Used Similarity Measure: None		
Method	**Iteration Number**	**Execution Time**
Gumbel	93	2.9972 secs
Laplace	2665	81.520 secs
Normal	1352	41.4493 secs
Random	7158	219.0264 secs

The best image quality of output image was for "Gumbel" comparing to other methods. It should be mentioned that the output image similarity for other algorithms was poor, and the similarity rate was under our acceptance range and our newly proposed method just improve the image as much as possible.

Neural Network Type: CNN		**Dataset:** CIFAR10
Used Similarity Measure: None		
Method	**Iteration Number**	**Execution Time**
Gumbel	40	1.4572 secs
Laplace	27	0.8755 secs
Normal	71	2.2524 secs
Random	95	3.1774 secs

The quality image was better preserved in other existing methods comparing to our newly proposed method (Gumbel).

Neural Network Type: FCNN		**Dataset:** CIFAR10
Used Similarity Measure: SSIM		
Method	**Iteration Number**	**Execution Time**
Gumbel	90	2.8768 secs
Laplace	1404	45.9374 secs
Normal	1417	45.9695 secs
Random	2565	84.7981 secs

The best image quality of output image was for "Gumbel" comparing to other methods and the same scenario with FCNN and CIFAR10 happened in this evaluation as well.

Neural Network Type: CNN		**Dataset:** CIFAR10
Used Similarity Measure: SSIM		
Method	**Iteration Number**	**Execution Time**
Gumbel	43	1.5740 secs
Laplace	30	1.0127 secs
Normal	70	2.2770 secs
Random	112	3.6563 secs

The state of preserving image quality is the same as the scenario in which we did not use SSIM.

Neural Network Type: FCNN		**Dataset:** CIFAR10
Used Similarity Measure: ISSM		
Method	**Iteration Number**	**Execution Time**
Gumbel	81	2.7712 secs
Laplace	2591	84.3289 secs
Normal	1959	63.8066 secs
Random	1650	53.4608 secs

The best image quality of output image was for "Gumbel" comparing to other methods.

Neural Network Type: CNN		**Dataset:** CIFAR10
Used Similarity Measure: ISSM		
Method	**Iteration Number**	**Execution Time**
Gumbel	50	1.7148 secs
Laplace	32	1.0926 secs
Normal	45	1.5525 secs
Random	107	3.5688 secs

The image quality of our proposed method (Gumbel) is the lowest among others.

We used the same settings for evaluating our methods which were based on genetics algorithm and the result is as followed:

Neural Network Type: FCNN		**Dataset:** CIFAR10
Method	**Accuracy to the target**	**Execution Time**
Random	0.33	98.0839 secs
Random + SSIM	0.36	163.1825 secs
Gumbel	0.39	96.6687 secs
Gumbel +SSIM	0.34	80.0205 secs

Neural Network Type: CNN		**Dataset:** CIFAR10
Method	**Accuracy to the target**	**Execution Time**
Random	0.32	287.1287 secs
Random + SSIM	0.32	258.4917 secs
Gumbel	0.74	193.1415 secs
Gumbel +SSIM	0.36	48.6840 secs

Neural Network Type: FCNN		**Dataset:** MNIST
Method	**Accuracy to the target**	**Execution Time**
Random	0.73	53.8739 secs
Gumbel	0.54	129.1987 secs
Gumbel +SSIM	0.78	262.9427 secs

Neural Network Type: CNN		**Dataset:** MNIST
Method	**Accuracy to the target**	**Execution Time**
Random	0.76	66.8253 secs
Gumbel	0.76	115.7534 secs
Gumbel +SSIM	0.69	71.7786 secs

In all the above test scenarios the quality and similarity of image comparing to the original one is almost the same and they are all in the acceptance range.

7. Conclusion

The focus of this research and evaluation was based on time execution while the output image quality should be untouched comparing to default methods. As a result, from the evaluation process it can be understood that for MNIST dataset:

- It’s simpler than CIFAR10 and that’s why FCNN performed much better than CNN
- Gumbel Fuzzer has the best performance in term of time and maintaining image quality
- ISSM made a huge positive impact on the performance of Gumbel Fuzzer
- In overall, due to simplicity of MNIST dataset, using SSIM has nearly the same result comparing to the raw version of fuzzers.
- Gumbel Genetic Adversarial Method cannot make a positive impact on the performance nor on the similarity percentage for this dataset due to its simplicity.
- If we need to use Gumbel Genetic Adversarial Method on CNN architecture, to get the highest possible performance, we need to apply the SSIM.

While for CIFAR10 dataset we can come to the conclusion that:

- It’s more complicated than MNIST and that’s why CNN performed much better than FCNN
- Best overall performance among fuzzers is for Laplace Random Fuzzer
- SSIM has a better impact on Gumbel Random and Laplace Random Fuzzers while ISSM done the same thing for Normal Random and Completely Random Fuzzers.
- Gumbel Genetic Adversarial Method giving us better results with CNN architecture.
- Gumbel Genetic Adversarial Method execution time is nearly less than half of the execution time of other methods whenever SSIM is applied, and the image similarity percentage is in the acceptable range.

- In overall, adding SSIM to Gumbel Genetic Adversarial Method is making a huge impact

8. REFERENCES

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow and R. Fergus, "Intriguing properties of neural networks," ICLR, abs/1312.6199, 2014.

[2] I. Goodfellow, N. Papernot, S. Huang, R. Dua, P. Abbeel and J. Clark, "Attacking machine learning with adversarial examples," 24 February 2017. [Online]. Available: https://openai.com/blog/adversarial-example-research/.

[3] Goodfellow, J. Shlens and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations (ICLR),*, 2015.

[4] C. Szegedy, W. Zaremba, I. Sutskeve, J. Bruna, D. Erhan, I. Goodfellow and R. Fergus, "Intriguing properties of neural networks.," *arXiv preprint arXiv:1312.6199,* 2013.

[5] I. J. Goodfellow, J. Shlens and C. Szegedy, "Explaining and harnessing adversarial examples.," *arXiv preprint arXiv:1412.6572,* 2014.

[6] A. Kurakin, I. Goodfellow and S. Bengio, "Adversarial examples in the physical world, arXiv:1607.02533," *2017.*

[7] S.-M. Moosavi-Dezfooli, A. Fawzi and P. Frossard, "DeepFool: a simple and accurate method to fool deep neural networks, arXiv:1511.04599," *2016.*

[8] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik and A. Swami, " The limitations of deep learning in adversarial settings.," *IEEE European symposium on security and privacy (EuroS&P) In 2016,* p. 372–387, 2016.

[9] J. Guo, Y. Jiang, Y. Zhao, Q. Chen and J. S. Dlfuzz, "Differential fuzzing testing of deep learning systems.," *In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering,* pp. 739-743, 2018.

[10] P. Zhang, Q. Dai and P. P. Cagfuzz, "Coverage-guided adversarial generative fuzzing testing of deep learning systems.," *arXiv preprint arXiv:1911.07931,* 2019.

[11] M. Alzantot, Y. Sharma, A. Elgohary, B.-J. Ho, M. Srivastava and K.-W. Chang, "Generating natural language adversarial examples," *arXiv preprint arXiv:1804.07998,* 2018.

[12] N. Narodytska, S. Kasiviswanathan, L. Ryzhyk, M. Sagiv and T. Walsh, "Verifying properties of binarized deep neural networks," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[13] N. Narodytska and S. P. Kasiviswanathan, "Simple black-box adversarial perturbations for deep networks," *arXiv preprint arXiv:1612.06299,* 2016.

[14] P. Rathore, A. Basak, S. H. Nistala and V. Runkana, "Untargeted, Targeted and Universal Adversarial Attacks and Defenses on Time Series," in *TCS Research Pune, India, Published at IJCNN*, 2020.

[15] H. F. Asmhan, H. M. Zahir and D. Cailin, "An information-theoretic measure for face recognition: Comparison with structural," 2014.

[16] M. A. Aljanabi, Z. M. Hussain, N. A. A. Shnain and S. F. Lu, "Design of a hybrid measure for image similarity: a statistical, algebraic, and information-theoretic approach," *European Journal of Remote Sensing, 2019/12/18, doi: 10.1080/22797254.2019.1628617,* .

[17] . Z. Wang, A. Bovik, H. Sheikh and E. Simoncelli, " Image quality assessment: From error visibility to structural similarity.," *IEEE Transactions on Image Processing, 13(4),* , p. 600–612, 2004.

[18] N. Hastings and B. Peacock, Statistical distributions. 4th Ed, Wiley, 2011.

[19] "Statistics - Gumbel Distribution," [Online]. Available: https://www.tutorialspoint.com/statistics/gumbel_distribution.htm.

[20] Stephanie, "Statistics How To," 7 July 2016. [Online]. Available: https://www.statisticshowto.com/gumbel-distribution.

[21] "Gumbel distribution," Wikipedia (2020), 2020. [Online]. Available: https://en.wikipedia.org/wiki/Gumbel_distribution.

[22] "Overview of the Gumbel, Logistic, Loglogistic and Gamma Distributions, Issue 56," October 2005. [Online]. Available: https://www.weibull.com/hotwire/issue56/relbasics56.htm.

[23] . W. Meeker and L. Escobar, in *Statistical Methods for Reliability Data*, New York, John Wiley & Sons, Inc, 1998.

[24] "NIST/SEMATECH e-Handbook of Statistical Methods," September 2005. [Online]. Available: http://www.itl.nist.gov/div898/handbook/.

[25] G. J. Saavedra, K. N. Rodhouse and D. M. Dunlav, "A Review of Machine Learning Applications in Fuzzing," in *Sandia National Laboratories*, Albuquerque, NM, USA, 2019.

[26] "Laplace_distribution," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Laplace_distribution.

[27] A. Lyon and B. J. Phil, "“Why are Normal Distributions Normal?”," 2014, p. 621–649.

[28] B. D. Gerber, J. A. Dubovsky, W. Kendall, H. Mevin and D. Roderick, "Supporting Information," *Data*, March 2015.

[29] F. C. e. al., "Keras," 2015. [Online]. Available: https://keras.io.

[30] M. Abadi et al., ""TensorFlow: A system for large-scale machinelearn- ing,” in Proc. OSDI, vol. 16," Savannah, GA, USA, 2016, p. 265–283.

[31] J.Schmidhuber, "Deep learning in neural networks," in *An overview of Neural Networks*, 2015, pp. 85-117.

[32] " Y. Kim, “Convolutional neural networks for sentence classifi-cation,” in Proc. Conf. Empir. Methods Nat. Lang. Process.(EMNLP), Doha, Qatar, 2014, pp. 1746–1751.".

[33] " A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenetclassifica- tion with deep convolutional neural networks,” inProc. Adv. Neural Inf. Process. Syst., 2012, pp. 1097–1105".