Deep Concolic: Concolic Testing for Deep Neural Networks

Davood Zaman Farsa[100752180], Feiyang Wang[100729909], and Muhammad Salik Nadeem[100727304]

Ontario Tech University, Oshawa, Ontario, Canada.

Abstract. Concolic testing is a combination of program execution and symbolic analysis in order to explore the execution paths of a software program. In this paper we are using a tool for testing and debugging Deep Neural Networks(DNNs) and it is based on concolic testing approach. Besides, we define a coverage criteria for DNNs and then perform concolic testing to increase test coverage. In the the experimental results we show the effectiveness of the concolic testing in reaching to a high coverage and finding adversarial examples.

Keywords: Concolic Testing · Deep Concolic · Software Testing · Deep Neural Networks · Adversarial Attacks.

1 Introduction

Nowadays, Deep Neural Networks (DNNs) have gained too much attraction for solving wide range of AI problems. However, the main concern is about the safety of using DNN in real-world systems, where faulty behaviors will put human lives in a very dangerous situation and also increase the risk of financial damage. So, testing is a main step in software industry and it helps to get meaningful information about the quality of the software. So far, researchers have not concentrated on works for testing DNNs systematically and they were only focusing on concrete execution or symbolic execution or gradient-based search individually and non of these approaches use the combination of the other. However, concolic testing uses the combination of concrete execution and symbolic analysis for exploring the execution path of the software.

For the final project of this course we decided to implement this paper [4], because it is possible to test the coverage criteria and also generate adversarial examples for making DNNs more robust and safe. In addition, we realized that there is a good potential opportunity to test and debug more models of DNNs with two dataset MNIST and Fashion-Mnist. By analysing the results of our experiments we found a path to enhance the performance of different Convolutional Neural Networks (CNNs) models to examine the capacity of deepconcolic as a white-box tool for DNNs. The rest of this paper is structured as follows: section 2 reviews the background behind this research. In section 3, the deepconcolic testing is explained in details. Section 4 provide details about the experimental results and, finally, in section 5 we conclude the contribution.

2 Background

With the rapidly increasing interest in Deep Learning and its implementations in many safety critical systems it is more important than ever to have tools that can rigorously test, validate and certify software that implements DNNs [2]. A feedforward DNN [1] consists of many neurons stacked on top of one another in many hidden layers in the system. Each neuron has a set of non-linear activation functions which are triggered based on the input valued it gets. These layers can be a dense (fully connected) layer, convolutional layer, flatten layer or max-pooling layer, with activation functions such as Sigmoid, Rectified Linear Unit (ReLU), Leaky ReLU, Softmax and etc.
As far as safety is concerned, DNNs notably show strong concerns in dealing with adversarial examples [5], whereby two enough inputs cause contradictory decision. In addition, a DNN is unsafe if there exist at least one adversarial example and as DNNs are considered as black boxes, so it id very difficult to understand their behavior by means of inspection. As an example, a self driving car that uses DNN as its model, if it recognise a driving sign incorrectly, then it will put the human lives in serious danger. Therefore, the model should be robust and accurate even with adversarial example.

3 The DeepConcolic Tool

Concolic testing has been applied a lot in software testing and it starts by using concrete input for executing the program, at the end of the concrete run, a new execution path is selected heuristically. This novel execution path is encoded symbolically and it yields a new concrete input. This process is repeated over and over until we reach a satisfactory level of structural coverage. The most impactful part that affects the performance of the concolic testing is the heuristic used to select the next execution path and more carefully designed heuristics will lead to achievement of better coverage. The architecture of DeepConcolic is presented in figure 1. The main work of DeepConcolic is to incrementally generate test suit to improve coverage by alternating between concrete execution and symbolic analysis and then return the coverage result. Therefore, it takes a DNN model and some raw test data as input, the reason for taking raw test data is to help this tool to find suitable nearby test cases easier. Then we have to define a test criterion which we use neuron coverage. Next step is preprocessing which formats the input data and tries to configure the concolic engine and the gradient ascent search engine (GA). Given an unsatisfied test requirement a, we identify a test input b based on our current test suit that is close to satisfy a according to the evaluations based on concrete execution and then we apply symbolic analysis to get a new test input c that satisfies a and we append c to the test suit. We keep repeating this process until finding a satisfactory level of coverage. At the last step, we determine whether the test suit includes adversarial examples (i.e. pairs of test cases that are close to each other with respect to a given distance measure but are different based on their classification labels) by passing the generated

test suit from the previous step to oracle database.
One of the key topics of our work is about the test coverage that we mentioned before. To be precise, in a software program there is a huge set of concrete execution paths [3] and, similarly, a DNN has a set of linear behaviours known as activation pattern which maps the set of hidden neurons to a set of {**true,false**}. The value "true" indicates that the ReLU of a neuron is activated and "false" otherwise. In fact, computing a test suit that covers the total activation patterns of a DNN is very difficult due to the large number of the neurons in DNNs. Thus, in this paper that we implemented [4], the authors have identified a subset of the activation patterns based on the coverage criteria, and try to generate test inputs that cover these activation patterns.
Now by providing a set of requirements on the test suit, we can define test coverage metric to show the percentage of the test requirements that are satisfied by the test cases T. Given a network N , a set R of test coverage requirements expressed as DR formulas, and a test suite T, the test coverage metric *M(R, T)* is as follows:

$$M(R,T) = \frac{|\{r \in R | T| = r\}|}{|R|} \tag{1}$$

The coverage metric in eq(1) is used as a proxy metric for the confidence in safety of the DNN under test. In our implementation we focused on the neuron coverage as the coverage criterion.

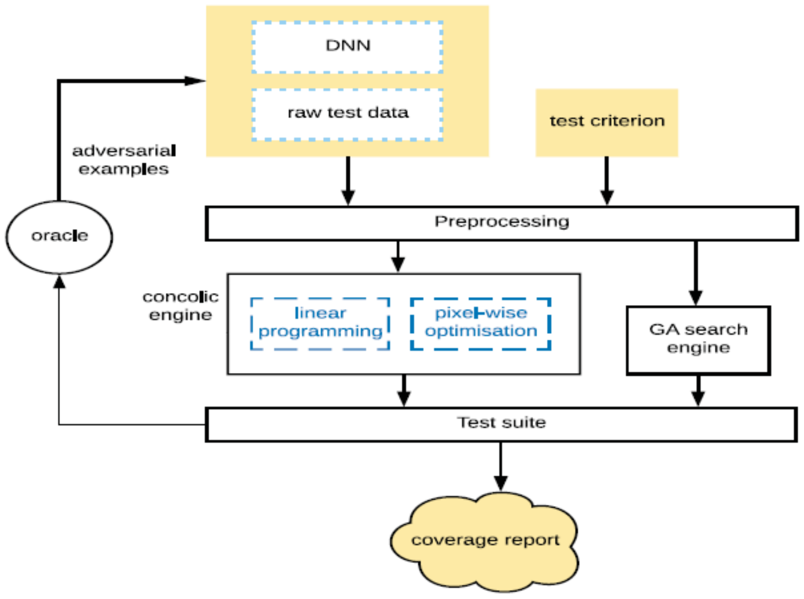


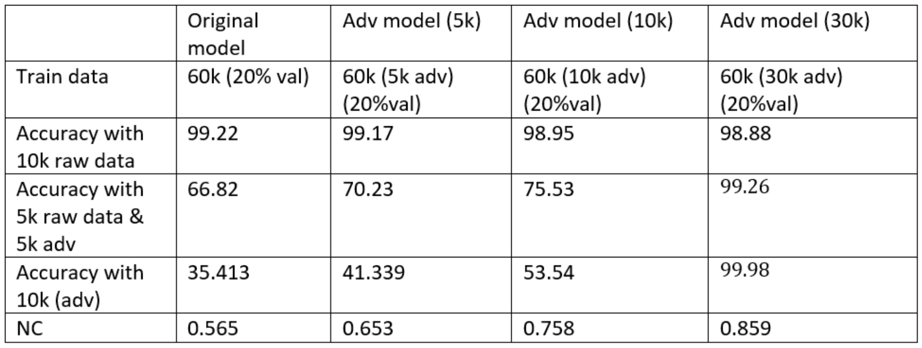
Fig. 1. The DeepConcolic Tool

4 Experimental Results

As the result of our implementation, we used the MNIST and fashion-MNIST datasets in our work. MNIST dataset contains 60000 images of size 28×28 pixels for training and 10000 images for testing and fasion-MNIST has 60000 training images of size 28×28 pixels and 10000 testing images. Both dataset are divided into 10 classes of images, MNIST has classes of numbers in range of 0 to 9 and fashion-MNIST has class labels of {T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot}.
First we trained the model based on the original data for both datasets. In the second experiment we substituted 5000 samples of the original data in each dataset with adversarial examples and for the third experiment we used 10000 adversarial examples instead of the original data and for the final one we used half of the original data and half from adversarial data. In addition to that, for testing the model with the 10000 images that each dataset has, we calculated the accuracy based on three different cases:

1. Accuracy with 10000 raw data
2. Accuracy with 5000 raw data and 5000 adversarial data
3. Accuracy with 10000 adversarial data

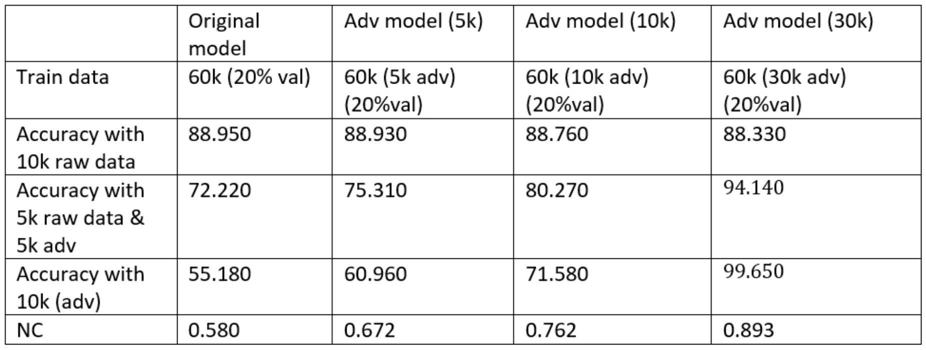
And we also calculated the neuron coverage in each of these experiments. The detailed result of our experiments is presented in Figure 2 and Figure 3.



	Original model	Adv model (5k)	Adv model (10k)	Adv model (30k)
Train data	60k (20% val)	60k (5k adv) (20%val)	60k (10k adv) (20%val)	60k (30k adv) (20%val)
Accuracy with 10k raw data	99.22	99.17	98.95	98.88
Accuracy with 5k raw data & 5k adv	66.82	70.23	75.53	99.26
Accuracy with 10k (adv)	35.413	41.339	53.54	99.98
NC	0.565	0.653	0.758	0.859

Fig. 2. Results for the MNIST data set

The results show a clear correlation with the robustness of the training process when using adversarial examples to train the model. The more adversarial examples are used in training the more robust the model and the more Neuron Coverage we get as a result. The original MNIST data set was more prone to adversarial attacks since it is a very simple data set and learn-able features are limited so the adversalial perturbations result in greater differences in the classification step. The Fashion-MNIST data set showed a bit more resilience to



	Original model	Adv model (5k)	Adv model (10k)	Adv model (30k)
Train data	60k (20% val)	60k (5k adv) (20%val)	60k (10k adv) (20%val)	60k (30k adv) (20%val)
Accuracy with 10k raw data	88.950	88.930	88.760	88.330
Accuracy with 5k raw data & 5k adv	72.220	75.310	80.270	94.140
Accuracy with 10k (adv)	55.180	60.960	71.580	99.650
NC	0.580	0.672	0.762	0.893

Fig. 3. Results for the Fashion-MNIST data set

adversarial attacks as compared to MNIST hand writings. This could be because while Fashion-MNIST has the same resolution, the data has more features than the simple hand writing cases and the networks learns those features in more detail so the minor changes in the adversarial examples were not as effective in its case.

5 Conclusion

In this paper we implemented DeepConcolic on two dataset MNIST and fashion-MNIST. We experimented our implementation in various cases based on the raw data and adversarial examples and realised that training a DNN with adversarial examples will result in higher accuracy and robustness. However, there were some challenges that we faced during our implementation such as, the original experiments were run on a machine with 24 core Intel(R) Xenon(R) CPU E5-2620 and 2.4GHz and 125GB memory but the hardware that we had was 2 GPU servers with 4 Titan V100 GPUs having 32GB memory. So, we had to change the CPU based code to run on the GPU and based on our experiments. Our results clearly show that training deep learning models with adversarial examples makes them more robust and secure to adversarial attacks and it also results in greater coverage in terms of the Neuron Coverage criterion as described in the paper.

References

1. Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1. Citeseer, 2017.
2. Xiaowei Huang, Daniel Kroening, Marta Kwiatkowska, Wenjie Ruan, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. Safety and trustworthiness of deep neural networks: A survey. *arXiv preprint arXiv:1812.08342*, 2018.
3. Youcheng Sun, Xiaowei Huang, and Daniel Kroening. Testing deep neural networks. *arXiv preprint arXiv:1803.04792*, 2018.

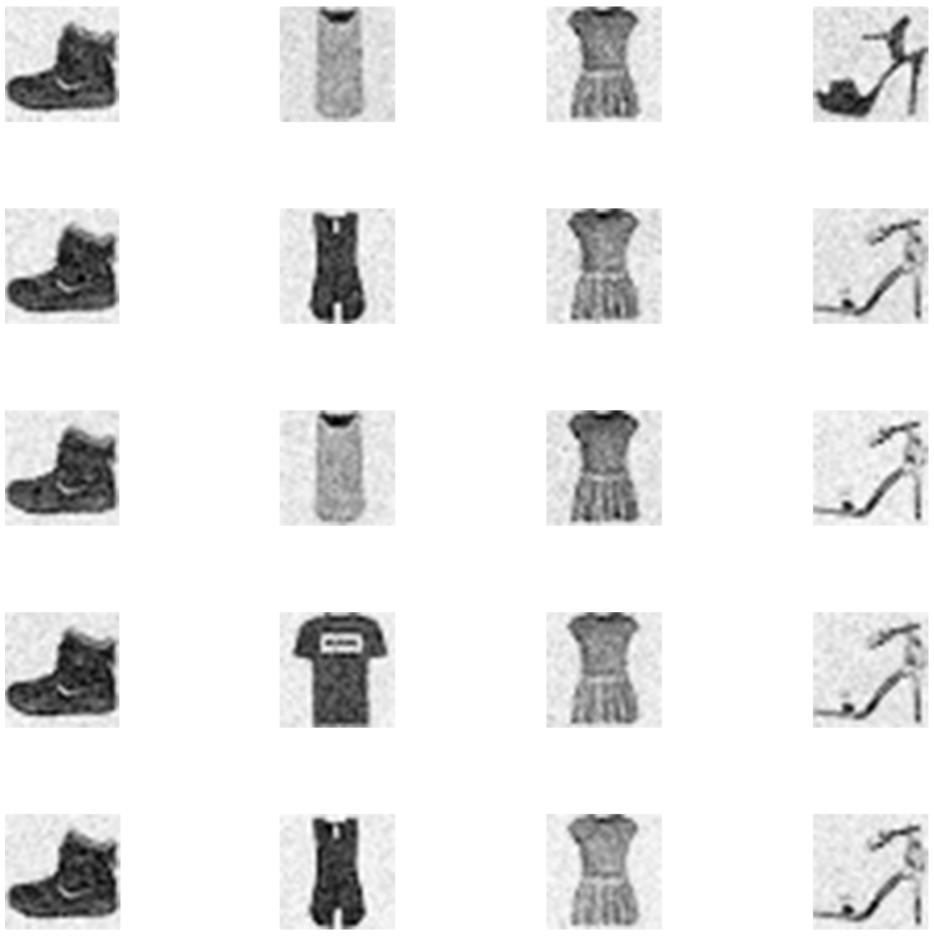


Fig. 4. Adversarial examples for the Fashion-MNIST data set

4. Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. Concolic testing for deep neural networks. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 109–119. ACM, 2018.
5. Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.