DeepXplore: Automated Whitebox Testing of Deep Learning Systems

Kexin Pei, Yinzhi Cao, Junfeng Yang, Suman Jana
SOSP 2017

Table of Contents

Table of Contents

Introduction

DL systems, despite their impressive capabilities, often demonstrate unexpected or incorrect behaviors in corner cases for several reasons such as biased training data, overfitting, and underfitting of the models.

Existing DL testing depends heavily on manually labeled data and therefore often fails to expose erroneous behaviors for rare inputs.

They design, implement, and evaluate **DeepXplore**, the first whitebox framework for systematically testing real-world DL systems.

Introduction

They address two main problems:

- Generating inputs that trigger different parts of a DL system's logic.
- Identifying incorrect behaviors of DL systems without manual effort.

First, they introduce **neuron coverage**. At a high level, neuron coverage of DL systems is similar to code coverage of traditional systems.

However, code coverage itself is not a good metric for estimating coverage of DL systems.

Even a single randomly picked test input was able to achieve 100% code coverage while the neuron coverage was less than 10%.

Introduction

Next, they show how multiple DL systems with similar functionality (e.g., self-driving cars by Google, Tesla, and GM) can be used as **cross-referencing oracles** to identify erroneous corner cases without manual checks.

For example, if one selfriving car decides to turn left while others turn right for the same input, one of them is likely to be incorrect. (differential testing)

Introduction

Finally, they demonstrate how the problem of generating test inputs that maximize neuron coverage of a DL system while also exposing as many differential behaviors (i.e., differences between multiple similar DL systems) as possible can be formulated as a joint optimization problem.

They design, implement, and evaluate **DeepXplore**, the first efficient **whitebox testing framework** for large-scale DL systems.

Table of Contents

DL Systems

We define a DL system to be any software system that includes at least one Deep Neural Network (DNN) component.

Note that some DL systems might comprise solely of DNNs (e.g., self-driving car DNNs predicting steering angles without any manual rules) while others may have some DNN components interacting with other traditional software to produce the final output.

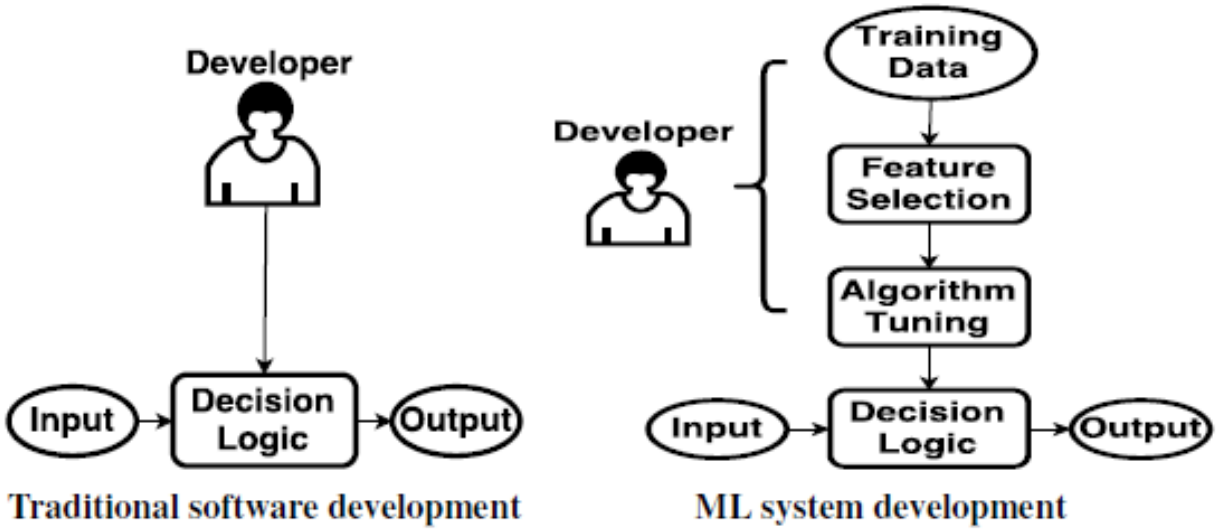


Figure: Comparison between traditional and ML system development processes.

DNN Architecture

DNNs are inspired by human brains with millions of interconnected neurons.

A DNN usually has at least three (often more) layers: one input, one output, and one or more hidden layers.

DNNs can be trained using different training algorithms, but gradient descent using backpropagation is by far the most popular training algorithm for DNNs.

DNN Architecture

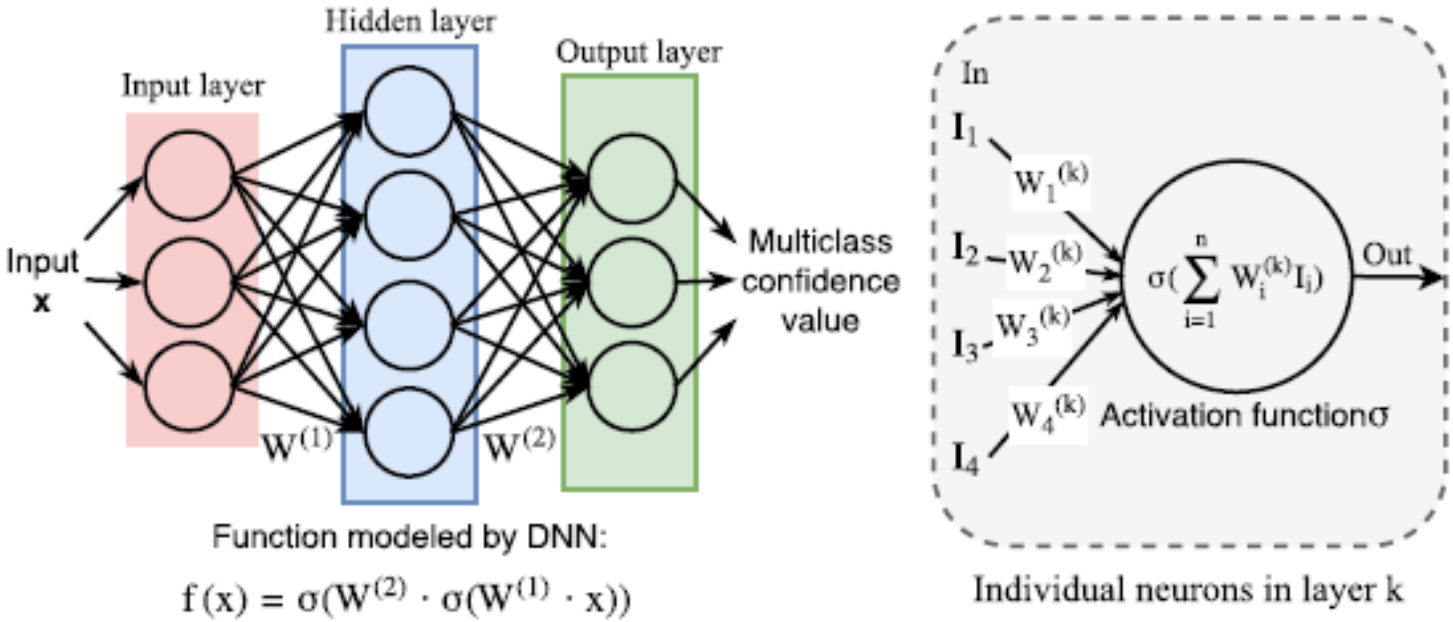


Figure: A simple DNN and the computations performed by each of its neurons.

Limitations of Existing DNN Testing

Expensive Labeling Effort

Existing DNN testing techniques require prohibitively expensive human effort to provide correct labels/actions for a target task (e.g., self-driving a car, image classification, and malware detection).

For complex and high-dimensional real-world inputs, human beings, even domain experts, often have difficulty in efficiently performing a task correctly for a large dataset.

Limitations of Existing DNN Testing

Low Test Coverage

None of the existing DNN testing schemes even try to cover different rules of the DNN.

Therefore, the test inputs often fail to uncover different erroneous behaviors of a DNN.

Limitations of Existing DNN Testing

Problems with low-coverage DNN tests.

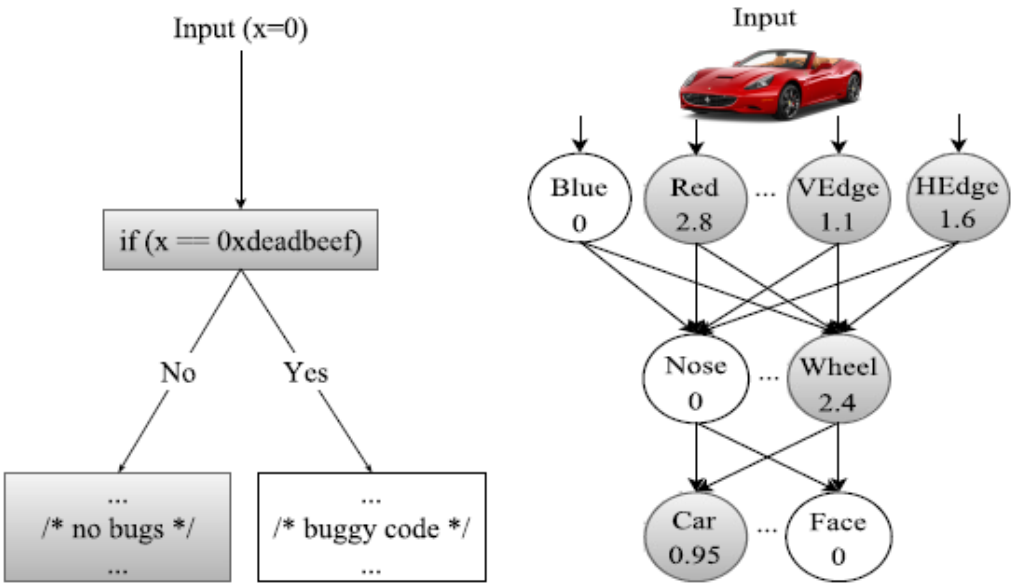


Figure: Comparison between program flows of a traditional program and a neural network. The nodes in gray denote the corresponding basic blocks or neurons that participated while processing an input.

The figure shows the similarity between traditional software and DNNs.

Limitations of Existing DNN Testing

Problems with low-coverage DNN tests

Of course, unlike traditional software, DNNs do not have explicit branches but a neuron's influence on the downstream neurons decreases as the neuron's output value gets lower.

Note that randomly picked inputs are highly unlikely to set high output values for the unlikely combination of neurons.

For example, if an image causes neurons labeled as "**Nose**" and "**Red**" to produce high output values and the DNN misclassifies the input image as a car, such a behavior will never be seen during regular testing as the chances of an image containing a red nose (*e.g., a picture of a clown*) is very small.

Table of Contents

Overview

Workflow

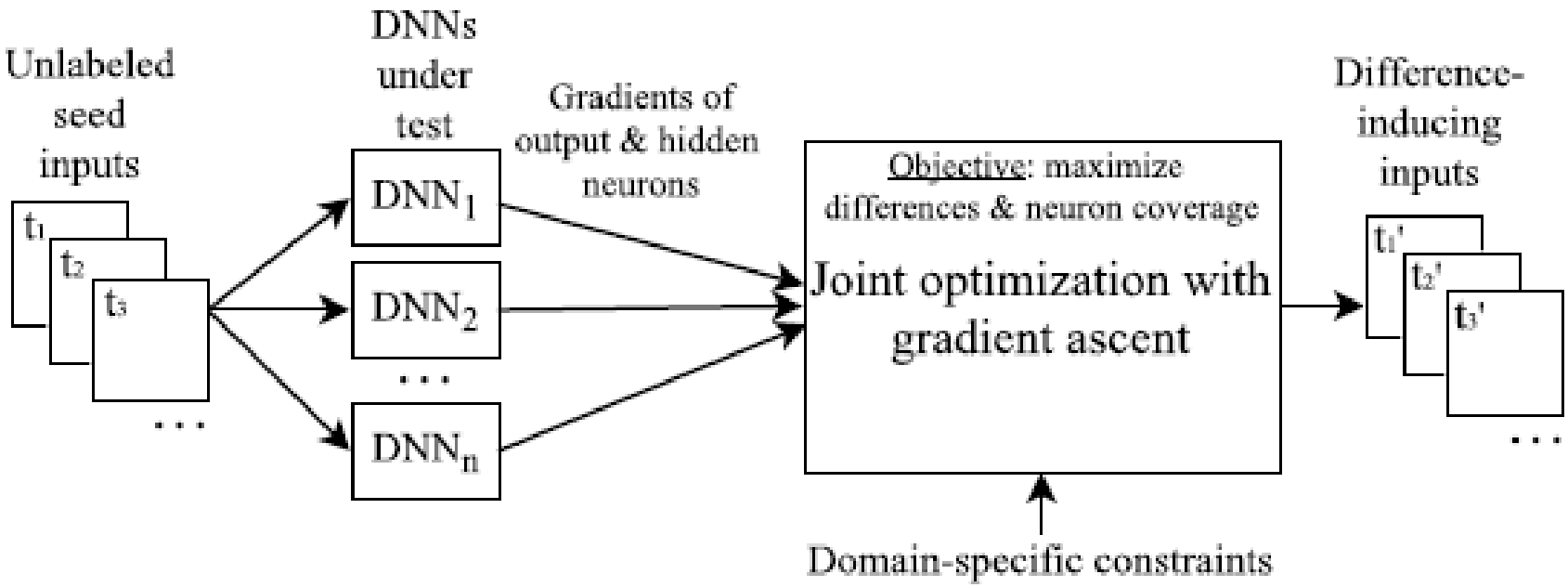
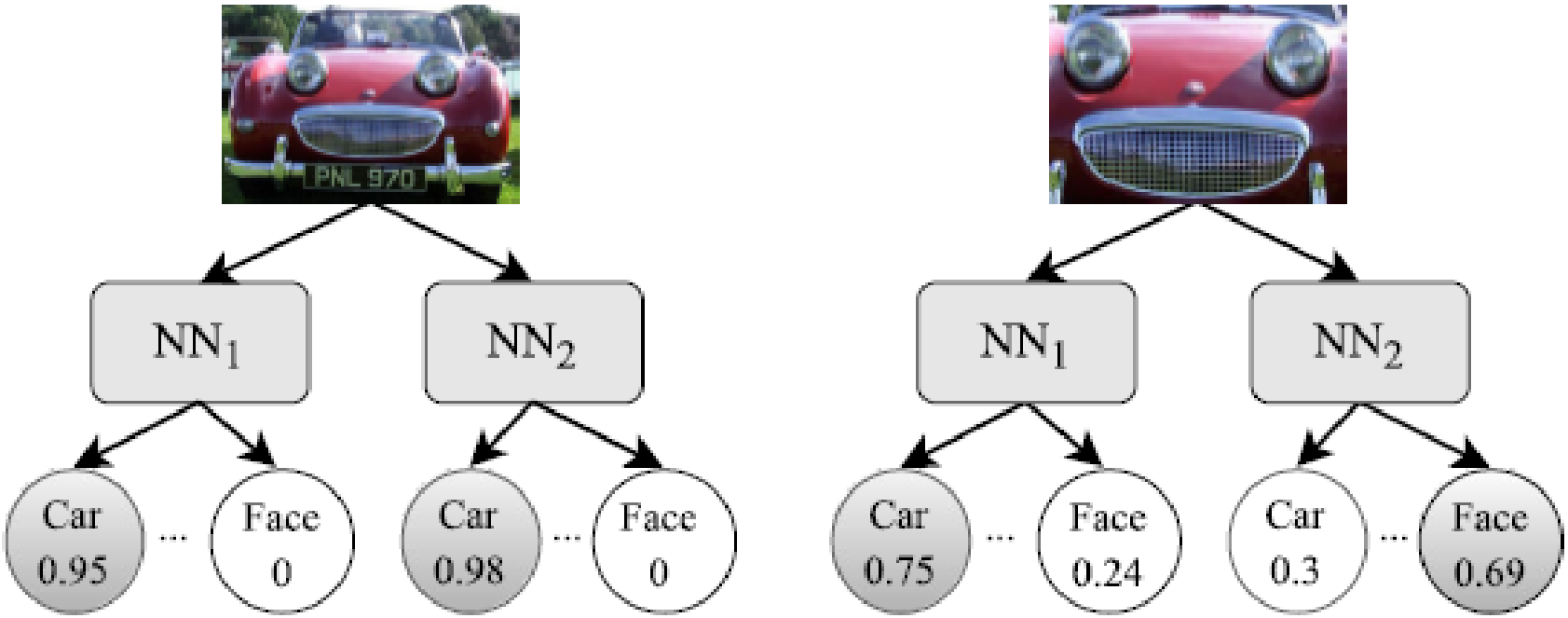


Figure: DeepXplore workflow.

DeepXplore solves a *joint optimization problem* that maximizes both **differential behaviors** and **neuron coverage**.

Overview

A working example



(a) DNNs produce same output (b) DNNs produce different output

Figure: Inputs inducing different behaviors in two similar DNNs.

Consider that we have two DNNs to test—both perform similar tasks, i.e., classifying images into cars or faces, but they are trained independently with different datasets and parameters. Therefore, the DNNs will learn similar but slightly different classification rules.

The joint optimization algorithm will iteratively perform a gradient ascent to find a modified input that satisfies all of the goals described.

Overview

A working example

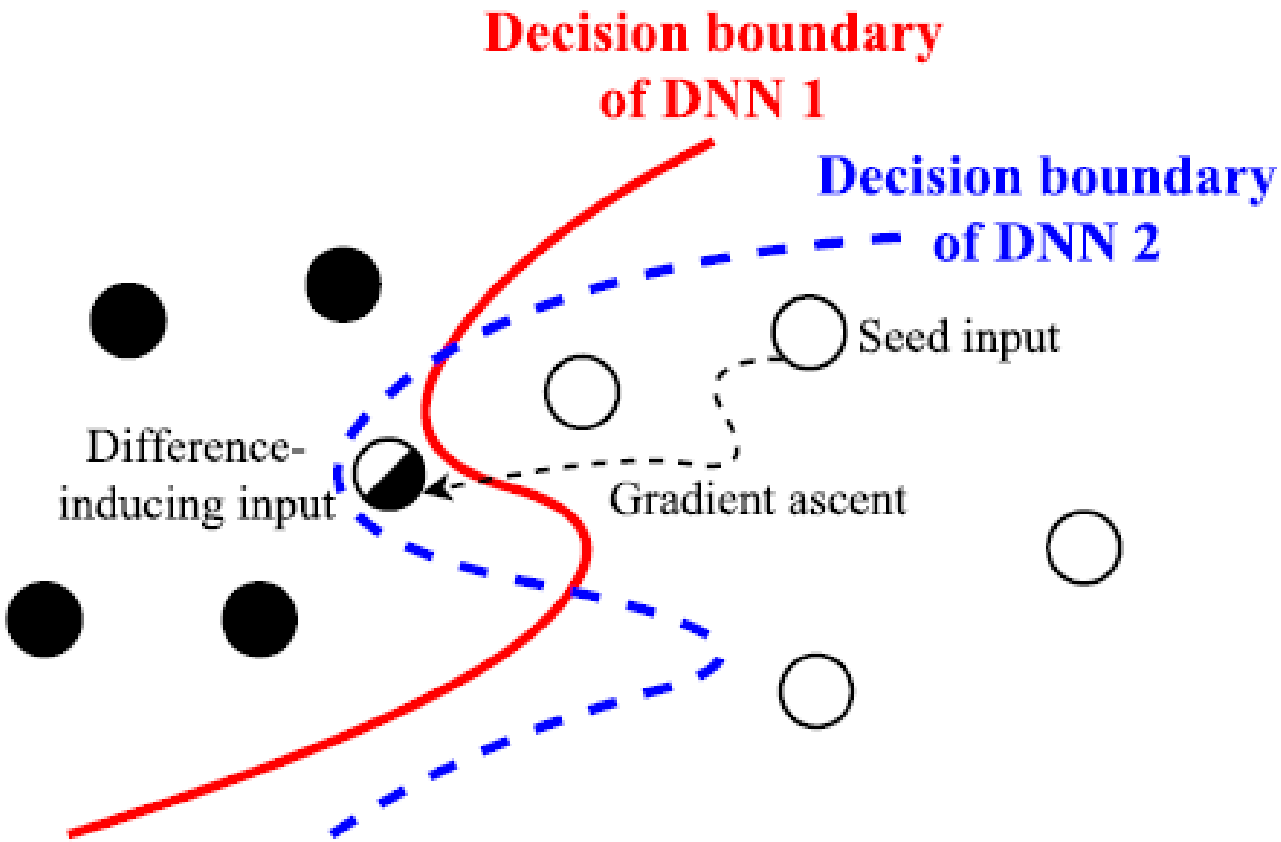


Figure: Gradient ascent starting from a seed input and gradually finding the difference-inducing test inputs.

Table of Contents

Neuron coverage: Neuron coverage of a set of test inputs as the ratio of the number of unique activated neurons for all test inputs and the total number of neurons in the DNN.

$$NCov(T, \mathbf{x}) = \frac{|\{n | \forall \mathbf{x} \in T, out(n, \mathbf{x}) > t\}|}{|N|}$$

$N = \{n_1, n_2, \dots\}$: all neurons of a DNN.
$T = \{x_1, x_2, \dots\}$: all test inputs.
$out(n, x)$: a function that returns the output value of neuron n in the DNN for a given test input x.

Gradient: The parametric function performed by a neuron can be represented as $y = f(\theta, x)$ where f is a function.

The gradient of $f(\theta, x)$ with respect to input x can be defined as:

$$G = \nabla_x f(\theta, x) = \frac{\partial y}{\partial x}$$

θ: parameters of DNN.
x: test input of DNN.

Methodology

DeepXplore Algorithm

They define test generation process as an optimization problem, thus it can be solved efficiently using gradient ascent.

Figure: Test input generation via joint optimization

Input: **seed_set** ← unlabeled inputs as the seeds
dnns ← multiple DNNs under test
λ_1 ← parameter to balance output differences of DNNs (Equation 2)
λ_2 ← parameter to balance coverage and differential behavior
s ← step size in gradient ascent
t ← threshold for determining if a neuron is activated
p ← desired neuron coverage
cov_tracker ← tracks which neurons have been activated

```
/* main procedure */
gen_test := empty set
for cycle(x ∈ seed_set) do // infinitely cycling through seed_set
    /* all dnns should classify the seed input to the same class */
    c = dnns[0].predict(x)
    d = randomly select one dnn from dnns
    while True do
        obj1 = COMPUTE_OBJ1(x, d, c, dnns, λ1)
        obj2 = COMPUTE_OBJ2(x, dnns, cov_tracker)
        obj = obj1 + λ2 · obj2
        grad = ∂obj / ∂x
        /*apply domain specific constraints to gradient*/
        grad = DOMAIN_CONSTRNTS(grad)
        x = x + s · grad //gradient ascent
        if d.predict(x) ≠ (dnns-d).predict(x) then
            /* dnns predict x differently */
            gen_test.add(x)
            update cov_tracker
            break
    if DESIRED_COVERAGE_ACHVD(cov_tracker) then
        return gen_test
/* utility functions for computing obj1 and obj2 */
procedure COMPUTE_OBJ1(x, d, c, dnns, λ1)
    rest = dnns - d
    loss1 := 0
    for dnn in rest do
        loss1 += dnn_c(x) //confidence score of x being in class c
    loss2 := d_c(x) //d's output confidence score of x being in class c
    return (loss1 - λ1·loss2)
procedure COMPUTE_OBJ2(x, dnns, cov_tracker)
    loss := 0
    for dnn ∈ dnns do
        select a neuron n inactivated so far using cov_tracker
        loss += n(x) //the neuron n's output when x is the dnn's input
    return loss
```

Maximizing differential behaviors: The first objective of the optimization problem is to generate test inputs that can induce different behaviors in the tested DNNs.
Suppose we have n DNNs:

$$F_{k \in 1 \ldots n} : x \rightarrow y$$

where
F_k: function modeled by the k-th neural network.
x: the input
y: output class probability vectors.

Let $F_k(x)[c]$ be the class probability that F_k predicts x to be c.

They maximize the following objective function:

$$obj_1(x) = \sum_{k \neq j} F_k(x)[c] - \lambda_1 \cdot F_j(x)[c]$$

λ_1: a parameter to balance the objective terms between the DNNs.
obj_1 can be maximized with gradient ascent.

Maximizing neuron coverage: The second objective is to generate inputs that maximize neuron coverage.
They want to maximize

$$obj_2(x) = f_n(x)$$

Such that $f_n(x) > t$

t: the neuron activation threshold.
$f_n(x)$: the function modeled by neuron n that takes x as input and produce the output of neuron n.

Joint optimization: They jointly maximize obj_1 and f_n described above and maximize the following function:

$$obj_{joint} = (\sum_{i \neq j} F_i(x)[c] - \lambda_1 \cdot F_j(x)[c]) + \lambda_2 \cdot f_n(x)$$

λ_2: a parameter for balancing between the two objectives of the joint optimization process
n: the inactivated neuron that we randomly pick at each iteration

Domain-specific constraints: One important aspect of the optimization process is that the generated test inputs need to satisfy several domain-specific constraints to be realistic.

They designed a simple rule-based method to ensure that the generated tests satisfy the custom domain-specific constraints.

Hyperparameters in the algorithm:
λ_1: Larger λ_1 puts higher priority on lowering the prediction value/confidence of a particular DNN while smaller λ_1 puts more weight on maintaining the other DNNs' predictions.

λ_2: Larger λ_2 focuses more on covering different neurons while smaller λ_2 generates more difference-inducing test inputs.

s: Larger s may lead to oscillation around the local optimum while smaller s may need more iterations to reach the objective.

t: Finding inputs that activate a neuron become increasingly harder as t increases.

Table of Contents

Implementation

Their code is built on TensorFlow/Keras but does not require any modifications to these frameworks.

Their experiments were run on a Linux laptop running Ubuntu 16.04 (one Intel i7-6700HQ 2.60GHz processor with 4 cores, 16GB of memory, and a NVIDIA GTX 1070 GPU).

Table of Contents

Evaluation Setup

Test Datasets and DNNs

They evaluate DeepXplore on three DNNs for each dataset (i.e., a total of fifteen trained DNNs).

- **MNIST**: large handwritten digit dataset containing 28x28 pixel images with class labels from 0 to 9. (60000 training and 10000 testing).
- **ImageNet**: large image dataset with over 10000000 hand-annotated images that are crowdsourced and labeled manually.
- **Driving**: Udacity self-driving car challenge dataset that contains images captured by a camera of a driving car and the simultaneous steering wheel angle applied by the human driver for each image. (101396 training and 5614 testing).
- **Contagio/VirusTotal**: dataset containing different benign and malicious PDF documents. (5000 + 12205 training -Contagio-, 5000 + 5000 testing -VirusTotal-. 135 static features from PDFrate)
- **Drebin**: dataset with 129013 Android applications among which 123453 are benign and 5560 are malicious. There is a total of 545333

Evaluation Setup

Test Datasets and DNNs

Figure: Details of the DNNs and datasets used to evaluate DeepXplore

Dataset	Dataset Description	DNN Description	DNN Name	# of Neurons	Architecture	Reported Acc.	Our Acc.
MNIST	Hand-written digits	LeNet variations	MNI_C1	52	LeNet-1, LeCun et al. [40, 42]	98.3%	98.33%
			MNI_C2	148	LeNet-4, LeCun et al. [40, 42]	98.9%	98.59%
			MNI_C3	268	LeNet-5, LeCun et al. [40, 42]	99.05%	98.96%
Imagenet	General images	State-of-the-art image classifiers from ILSVRC	IMG_C1	14,888	VGG-16, Simonyan et al. [66]	92.6%**	92.6%**
			IMG_C2	16,168	VGG-19, Simonyan et al. [66]	92.7%**	92.7%**
			IMG_C3	94,059	ResNet50, He et al. [31]	96.43%**	96.43%**
Driving	Driving video frames	Nvidia DAVE self-driving systems	DRV_C1	1,560	Dave-orig [8], Bojarski et al. [10]	N/A	99.91%#
			DRV_C2	1,560	Dave-norminit [78]	N/A	99.94%#
			DRV_C3	844	Dave-dropout [18]	N/A	99.96%#
Contagio/Virustotal	PDFs	PDF malware detectors	PDF_C1	402	<200, 200>+	98.5%−	96.15%
			PDF_C2	602	<200, 200, 200>+	98.5%−	96.25%
			PDF_C3	802	<200, 200, 200, 200>+	98.5%−	96.47%
Drebin	Android apps	Android app malware detectors	APP_C1	402	<200, 200>+, Grosse et al. [29]	98.92%	98.6%
			APP_C2	102	<50, 50>+, Grosse et al. [29]	96.79%	96.82%
			APP_C3	212	<200, 10>+, Grosse et al. [29]	92.97%	92.66%

** top-5 test accuracy; we exactly match the reported performance as we use the pretrained networks
# we report 1-MSE (Mean Squared Error) as the accuracy because steering angle is a continuous value
+ <x,y,...> denotes three hidden layers with x neurons in first layer, y neurons in second layer and so on
- accuracy using SVM as reported by Šrndic et al. [79]

Evaluation Setup

Domain Specific Constraints

Image Constraints: three different types of constraints for simulating different environment conditions of images:

1. lighting effects for simulating different intensities of lights,
2. occlusion by a single small rectangle for simulating an attacker potentially blocking some parts of a camera
3. occlusion by multiple tiny black rectangles for simulating effects of dirt on camera lens.

Evaluation Setup

Domain Specific Constraints

Different lighting conditions:

Occlusion with a single small rectangle:

Occlusion with multiple tiny black rectangles:

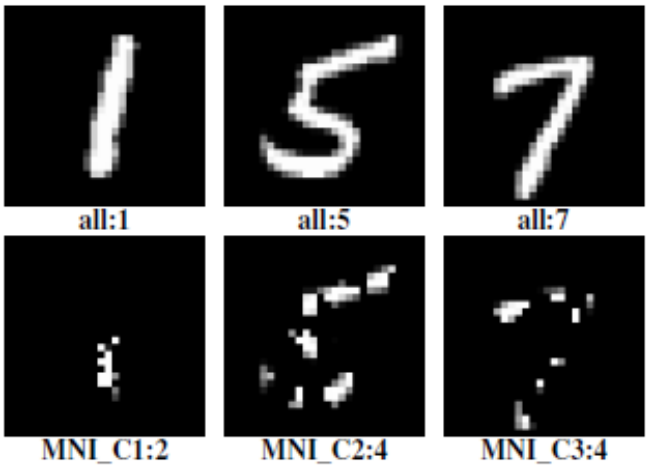


Table of Contents

Results

DNN name	Hyperparams (Algorithm 1)				# Differences Found
	λ_1	λ_2	s	t	
MNI_C1	1	0.1	10	0	1,073
MNI_C2					1,968
MNI_C3					827
IMG_C1	1	0.1	10	0	1,969
IMG_C2					1,976
IMG_C3					1,996
DRV_C1	1	0.1	10	0	1,720
DRV_C2					1,866
DRV_C3					1,930
PDF_C1	2	0.1	0.1	0	1,103
PDF_C2					789
PDF_C3					1,253
APP_C1	1	0.5	N/A	0	2,000
APP_C2					2,000
APP_C3					2,000

Figure: Number of difference-inducing inputs found by DeepXplore for each tested DNN obtained by randomly selecting 2,000 seeds from the corresponding test set for each run.

Results

input 1	**feature**	*feature:: bluetooth*	*activity:: .SmartAlertTerms*	*service_receiver:: .rrltpsi*
	before	0	0	0
	after	1	1	1
input 2	**feature**	*provider:: xclockprovider*	*permission:: CALL_PHONE*	*provider:: contentprovider*
	before	0	0	0
	after	1	1	1

Figure: The features added to the manifest file by DeepXplore for generating two sample malware inputs which Android app classifiers (Drebin) incorrectly mark as benign.

input 1	**feature**	*size*	*count_action*	*count_endobj*
	before	1	0	1
	after	34	21	20
input 2	**feature**	*size*	*count_font*	*author_num*
	before	1	0	10
	after	27	15	5

Figure: The top-3 most in(de)cremented features for generating two sample malware inputs which PDF classifiers incorrectly mark as benign.

Results

Benefits of Neuron Coverage

It has recently been shown that each neuron in a DNN tends to independently extract a specific feature of the input instead of collaborating with other neurons for feature extraction.

This finding intuitively explains why neuron coverage is a good metric for DNN testing comprehensiveness.

Results

Benefits of Neuron Coverage

Neuron coverage vs. code coverage: They set the threshold t in neuron coverage 0.75.

Dataset	Code Coverage			Neuron Coverage		
	C1	*C2*	*C3*	*C1*	*C2*	*C3*
MNIST	100%	100%	100%	32.7%	33.1%	25.7%
ImageNet	100%	100%	100%	1.5%	1.1%	0.3%
Driving	100%	100%	100%	2.5%	3.1%	3.9%
VirusTotal	100%	100%	100%	19.8%	17.3%	17.3%
Drebin	100%	100%	100%	16.8%	10%	28.6%

Figure: Comparison of code coverage and neuron coverage for 10 randomly selected inputs from the original test set of each DNN.

Results

Benefits of Neuron Coverage

Effect of neuron coverage on the difference-inducing inputs found by DeepXplore: They evaluate the effectiveness of neuron coverage at generating *diverse* difference-inducing inputs.

Exp. #	$\lambda_2 = 0$ (w/o neuron coverage)			$\lambda_2 = 1$ (with neuron coverage)		
	Avg. diversity	*NC*	*# Diffs*	*Avg. diversity*	*NC*	*# Diffs*
1	237.9	69.4%	871	283.3	70.6%	776
2	194.6	66.7%	789	253.2	67.8%	680
3	170.8	68.9%	734	182.7	70.2%	658

Figure: The increase in diversity (L1-distance) in the difference-inducing inputs found by DeepXplore while using neuron coverage as part of the optimization goal. This experiment uses 2,000 randomly picked seed inputs from the MNIST dataset. Higher values denote larger diversity. NC denotes the neuron coverage (with t = 0.25) achieved under each setting.

Results

Benefits of Neuron Coverage

They measure the *diversity* of the generated difference-inducing inputs in terms of averaged L1 distance between all difference-inducing inputs generated from the same seed and the original seed. The L1 distance calculates the sum of absolute differences of each pixel values between the generated image and the original one.

Also, the numbers of difference-inducing inputs generated with $\lambda_2 = 1$ are less than those for $\lambda_2 = 0$ as setting $\lambda_2 = 1$ causes DeepXplore to focus on finding diverse differences rather than simply increasing the number of differences with the same underlying root cause.

Results

Benefits of Neuron Coverage

Activation of neurons for different classes of inputs: Figure shows the results, which confirm our hypothesis that inputs coming from the same class share more activated neurons than those coming from different classes.

	Total neurons	*Avg. no. of activated neurons*	*Avg. overlap*
Diff. class	268	83.6	45.9
Same class	268	84.1	74.2

Figure: Average number of overlaps among activated neurons for a pair of inputs of the same class and different classes. Inputs of different classes tend to activate different neurons.

Results

Performance

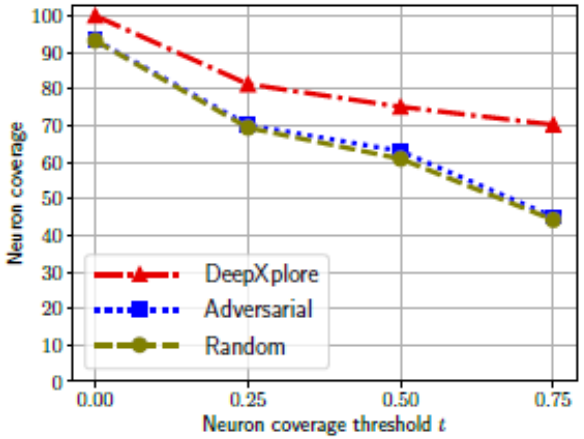
They evaluate DeepXplore's performance using two metrics: *neuron coverage of the generated tests* and *execution time for generating difference-inducing inputs*.

Neuron coverage: In this experiment, they compare the neuron coverage achieved by the same number of tests generated by three different approaches:

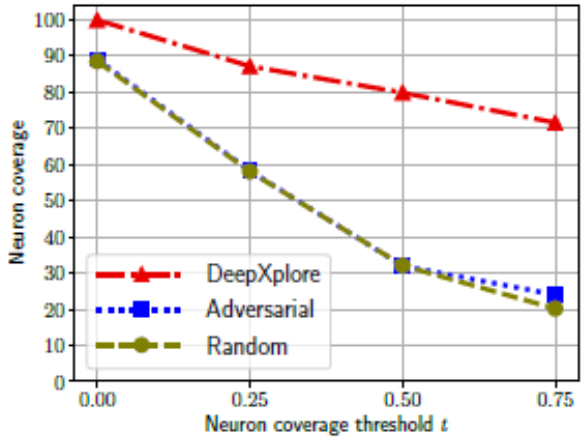
1. DeepXplore
2. adversarial testing
3. random selection from the original test set.

Results

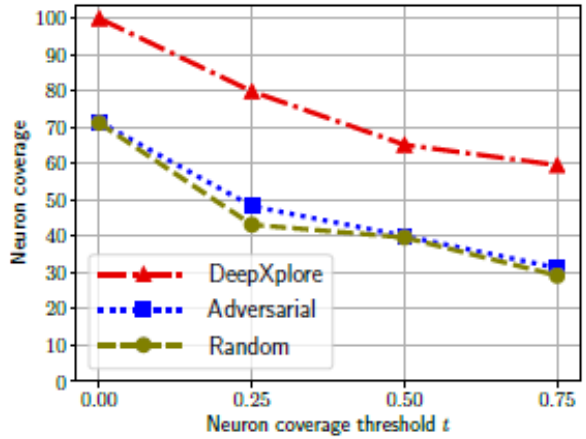
Performance



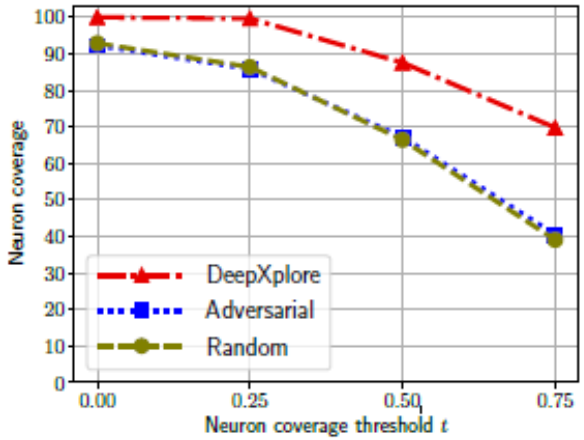
(a) MNIST



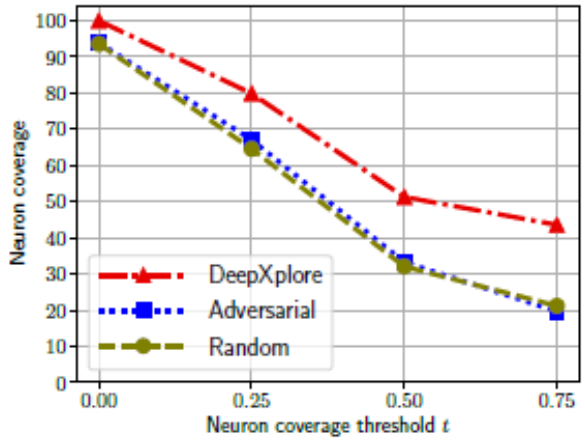
(b) ImageNet



(c) Driving



(d) VirusTotal



(e) Drebin

(1% of the original test set)

Execution time and number of seed inputs: They measure the execution time of DeepXplore to generate difference-inducing inputs with 100% neuron coverage for all the tested DNNs.

	C1	C2	C3	# seeds
MNIST	6.6 s	6.8 s	7.6 s	9
ImageNet	43.6 s	45.3 s	42.7 s	35
Driving	11.7 s	12.3 s	9.8 s	12
VirusTotal	31.1 s	29.7 s	23.2 s	6
Drebin	180.2 s	196.4 s	152.9 s	16

Figure: Averaged over 10 runs.

Different choices of hyperparameters: They evaluate how the choices of different hyperparameters influence DeepXplore's performance.

	s=0.01	*s=0.1*	*s=1*	*s=10*	*s=100*
MNIST	0.19 s	0.26 s	0.65 s	0.62 s	0.65 s
ImageNet	9.3 s	4.78 s	1.7 s	1.06 s	6.66 s
Driving	0.51 s	0.5 s	0.53 s	0.55 s	0.54 s
VirusTotal	0.17 s	0.16 s	0.21 s	0.21 s	0.22 s
Drebin	7.65 s	7.65 s	7.65 s	7.65 s	7.65 s

Figure: The variation in DeepXplore runtime (in seconds) while generating the first difference-inducing input for the tested DNNs with different step size choice. All numbers averaged over 10 runs. The fastest times for each dataset is highlighted in gray.

Results

Performance

	$\lambda_1 = 0.5$	$\lambda_1 = 1$	$\lambda_1 = 2$	$\lambda_1 = 3$
MNIST	0.28 s	0.25 s	0.2 s	0.17 s
ImageNet	1.38 s	1.26 s	1.21 s	1.72 s
Driving	0.62 s	0.59 s	0.57 s	0.58 s
VirusTotal	0.13 s	0.12 s	0.05 s	0.09 s
Drebin	6.4 s	5.8 s	6.12 s	7.5 s

Figure: The variation in DeepXplore runtime (in seconds) while generating the first difference-inducing input for the tested DNNs with different λ_1. Higher λ_1 values indicate prioritization of minimizing a DNNs' outputs over maximizing the outputs of other DNNs showing differential behavior. The fastest times for each dataset is highlighted in gray.

Results

Performance

	$\lambda_2 = 0.5$	$\lambda_2 = 1$	$\lambda_2 = 2$	$\lambda_2 = 3$
MNIST	0.19 s	0.22 s	0.23 s	0.26 s
ImageNet	1.55 s	1.6 s	1.62 s	1.85 s
Driving	0.56 s	0.61 s	0.67 s	0.74 s
VirusTotal	0.09 s	0.15 s	0.21 s	0.25 s
Drebin	6.14 s	6.75 s	6.82 s	6.83 s

Figure: The variation in DeepXplore runtime (in seconds) while generating the first difference-inducing input for the tested DNNs with different λ_2. Higher λ_2 values indicate higher priority for increasing coverage. All numbers averaged over 10 runs. The fastest times for each dataset is highlighted in gray.

Testing very similar models with DeepXplore: DeepXplore may fail to find any difference-inducing inputs within a reasonable time for some cases especially for DNNs with very similar decision boundaries.

They control three types of differences between two DNNs and measure the changes in iterations required to generate the first difference-inducing inputs in each case.

Training Samples	*# diff*	*0*	1	*100*	*1000*	*10000*
	# iter	-*	-*	616.1	503.7	256.9
Neurons per layer	*# diff*	*0*	1	*2*	*3*	*4*
	# iter	-*	69.6	53.9	33.1	18.7
Training Epochs	*# diff*	*0*	5	*10*	*20*	*40*
	# iter	-*	453.8	433.9	348.7	210

*- indicates timeout after 1000 iterations

Figure: Changes in the number of iterations DeepXplore takes, on average, to find the first difference inducing inputs as the type and numbers of differences between the test DNNs increase.

Results

Improving DNNs with DeepXplore

They demonstrate two additional applications of the error-inducing inputs generated by DeepXplore:

- augmenting training set and then improve DNN's accuracy
- detecting potentially corrupted training data.

Augmenting training data to improve accuracy:

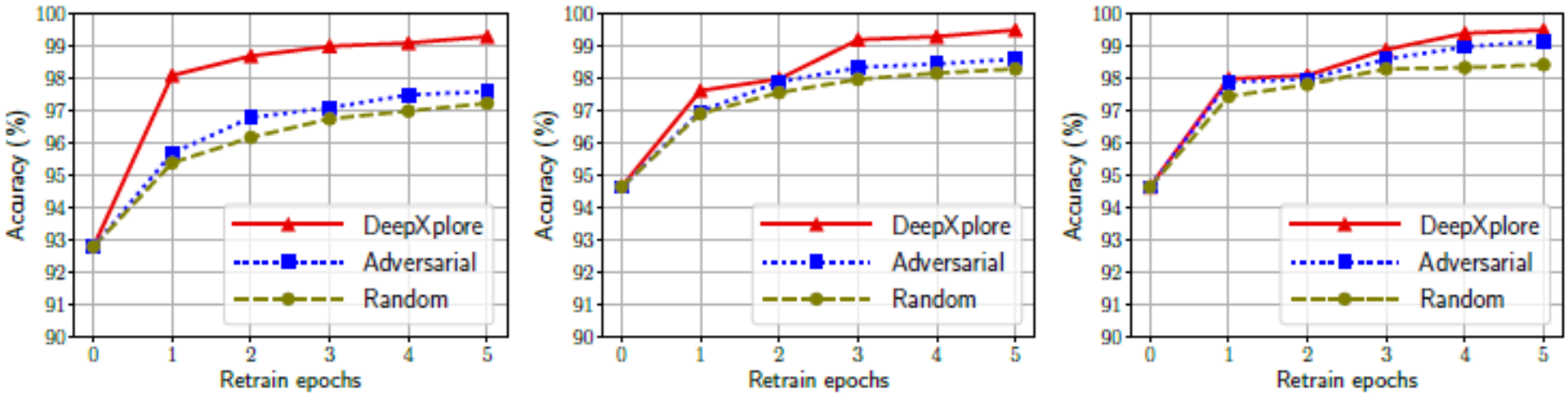


Figure: Improvement in accuracy of three LeNet DNNs when the training set is augmented with the same number of inputs generated by random selection ("random"), adversarial testing ("adversarial"), and DeepXplore.

Results

Improving DNNs with DeepXplore

Detecting training data pollution attack: tThey use wo LeNet-5 DNNs: one trained on 60, 000 hand-written digits from MNIST dataset and the other trained on an artificially polluted version of the same dataset where 30% of the images originally labeled as digit 9 are mislabeled as 1. We use DeepXplore to generate error-inducing inputs that are classified as the digit 9 and 1 by the unpolluted and polluted versions of the LeNet-5 DNN respectively. We then search for samples in the training set that are closest to the inputs generated by DeepXplore in terms of structural similarity and identify them as polluted data. Using this process, we are able to correctly identify 95.6% of the polluted samples.

Discussoins from Yoav Hollander

https://blog.foretellix.com/2017/06/06/deepxplore-and-new-ideas-for-verifying-ml-systems/