DeepXplore

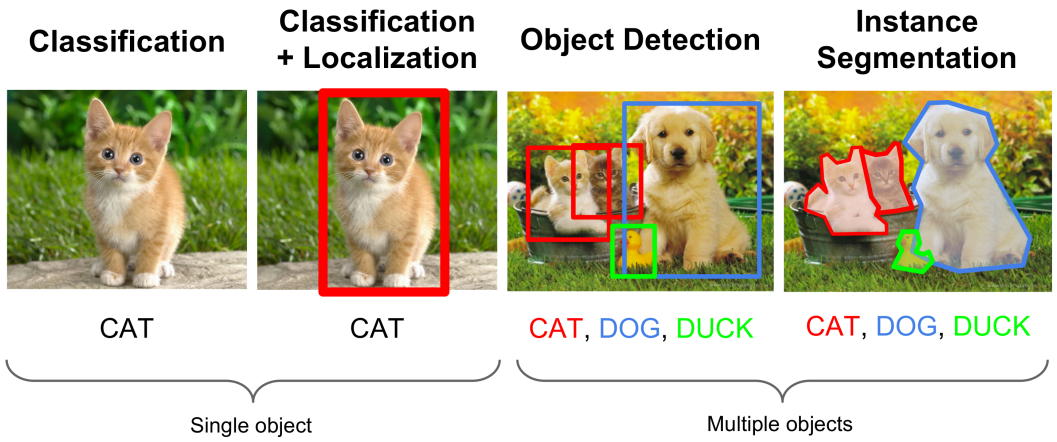
:Automated Whitebox Testing of Deep Learning Systems

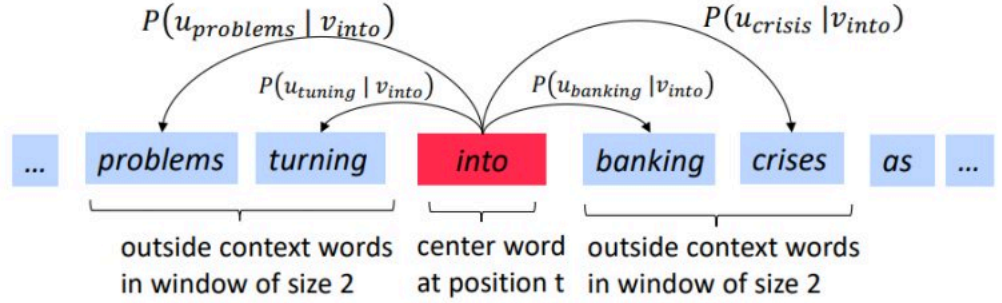
Kexin Pei, Yinzhi Cao, Junfeng Yang, Suman Jana

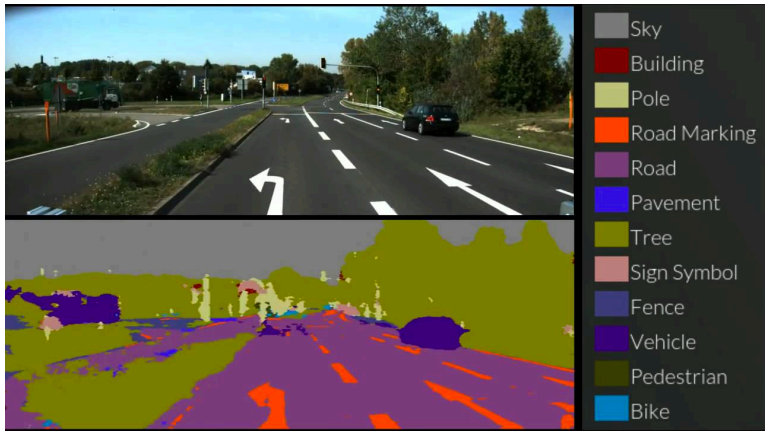
Hyewon Ryu

October 14, 2020 @ IS893

Nowadays… Deep Learning everywhere

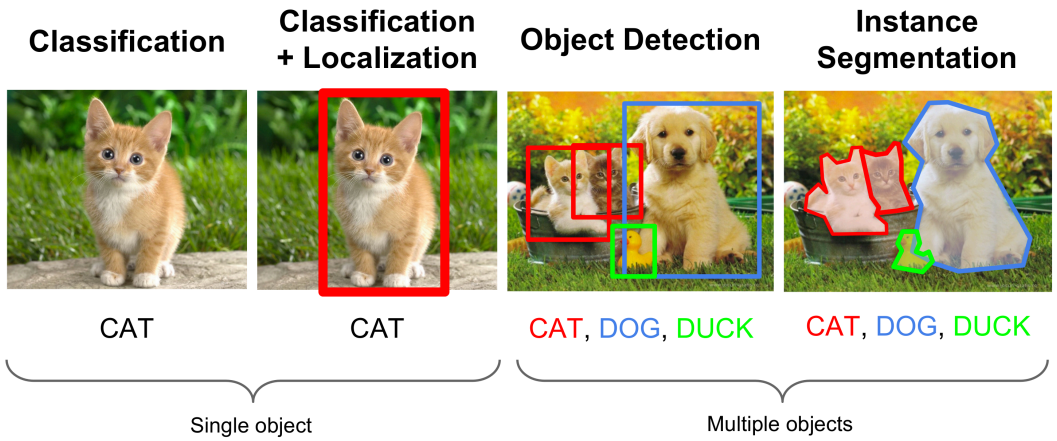


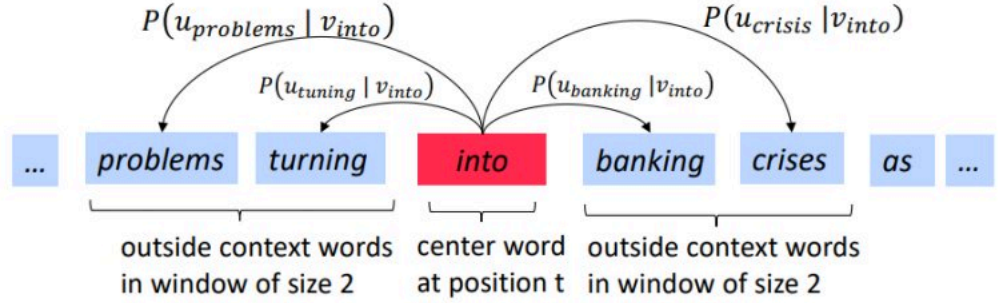




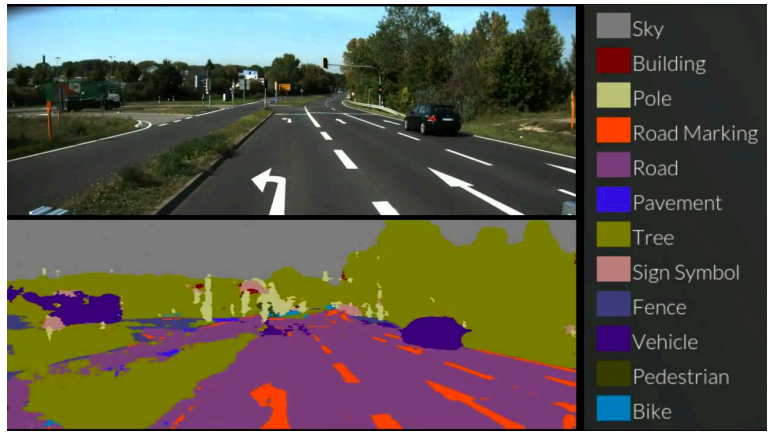
AlphaGo

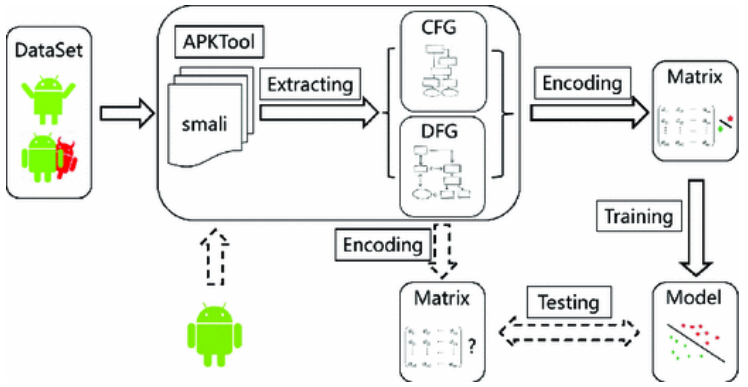
Nowadays… Deep Learning everywhere





Nowadays… Deep Learning everywhere

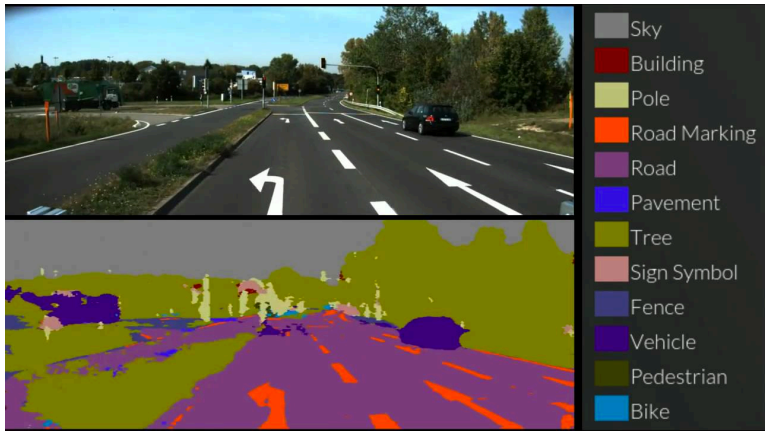


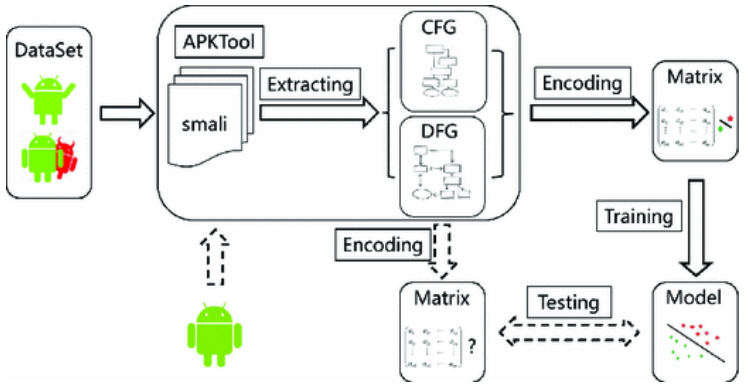


Nowadays… Deep Learning everywhere

Self-driving Uber car that hit and killed woman did not recognize that pedestrians jaywalk

https://www.nbcnews.com/tech/tech-news/self-driving-uber-car-hit-killed-woman-did-not-recognize-n1079281

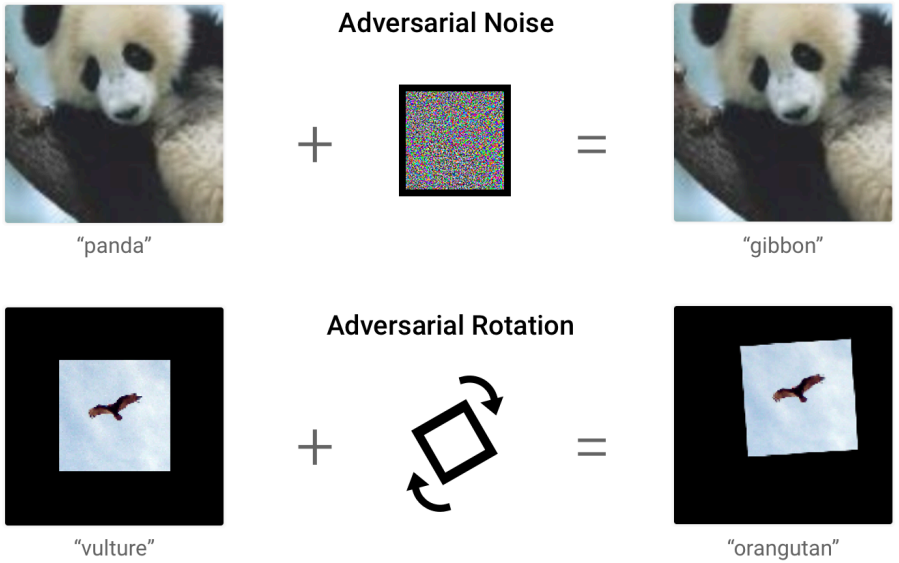




Testing DL until now

- measure accuracy with randomly chosen data

- Adversarial testing



Testing DL until now

- measure accuracy with randomly chosen data
- Adversarial testing

Problems

- Low coverage
  : tiny fraction of real world data (only subset of learned rule)
  adversarial testing is limited
  + how to measure coverage of DL system
- Expensive labeling effort
  : need manual labeling to enlarge data set

We need automated system that

Problems
- Low coverage
- Expensive labeling effort

- generate inputs
    trigger different parts of a DL system's logic
    uncover different types of erroneous behaviors

- identify erroneous behaviors without manual labeling

DeepXplore

:Automated Whitebox Testing of Deep Learning Systems

- introduce how to measure DL system coverage
  : neuron coverage
- avoid manual checking
  : cross-referencing of similar DNNs (like differential testing)
- automatically generate test input that
  induce erroneous behavior & activate more neurons
  by solving joint optimization problem.

Overview

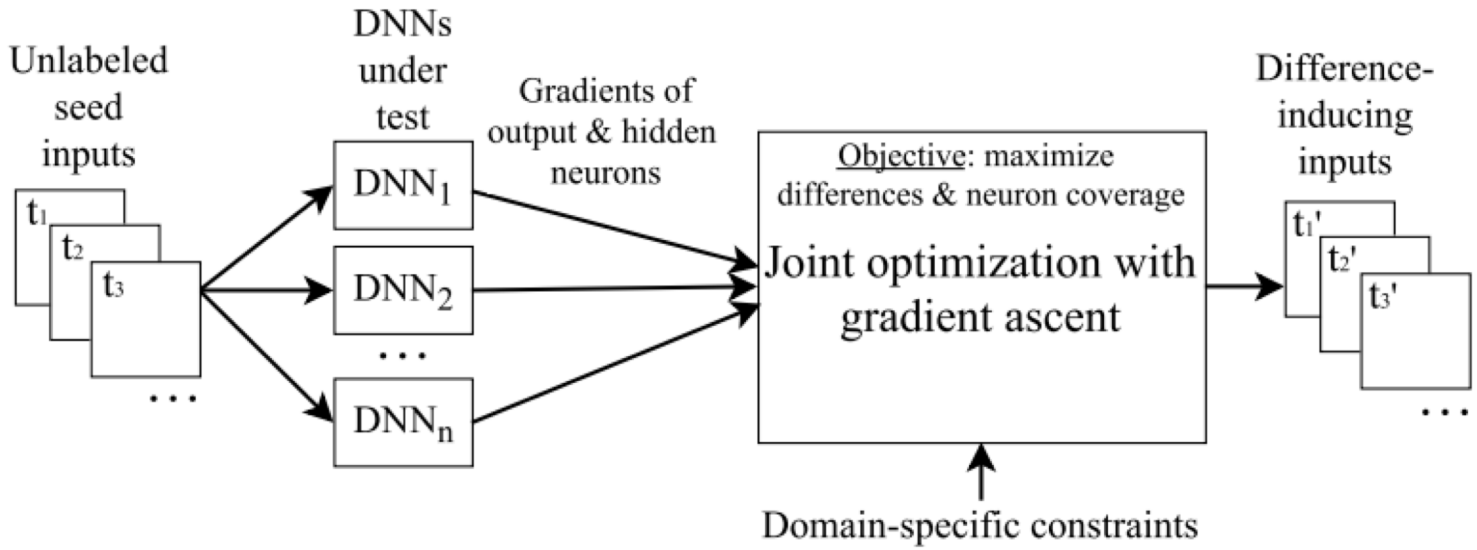


Figure 5: DeepXplore workflow.

- Neuron coverage: various type of corner case
- Cross-referencing: induce erroneous behavior

} joint optimization problem : diverse erroneous corner case

- Domain-specific constraint: make input valid
- Gradient ascent: efficiently solve optimization problem

Traditional vs DL system system

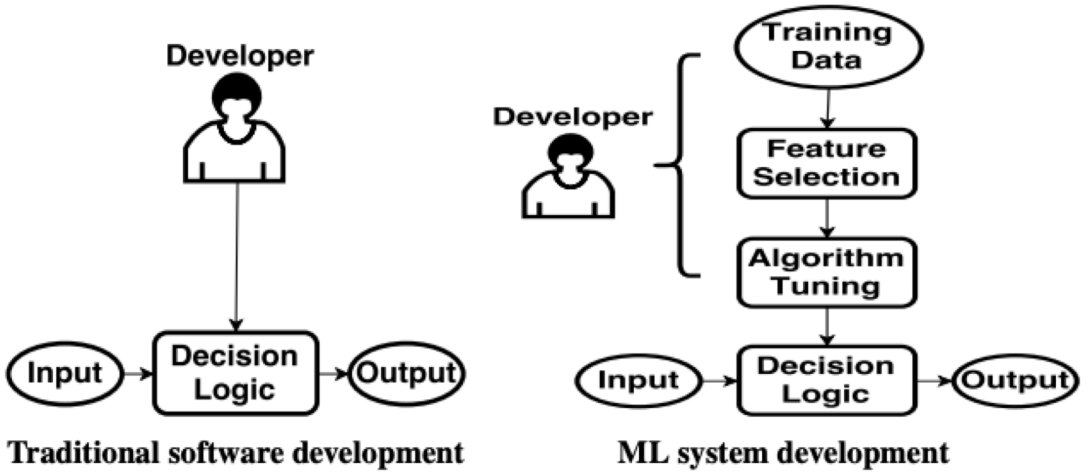


Figure 2: Comparison between traditional and ML system development processes.

Traditional	developers directly specify system logic
DL system	DNNs learn the rules developers indirectly influence (modifying data, feature, architecture)

Traditional vs DL system system

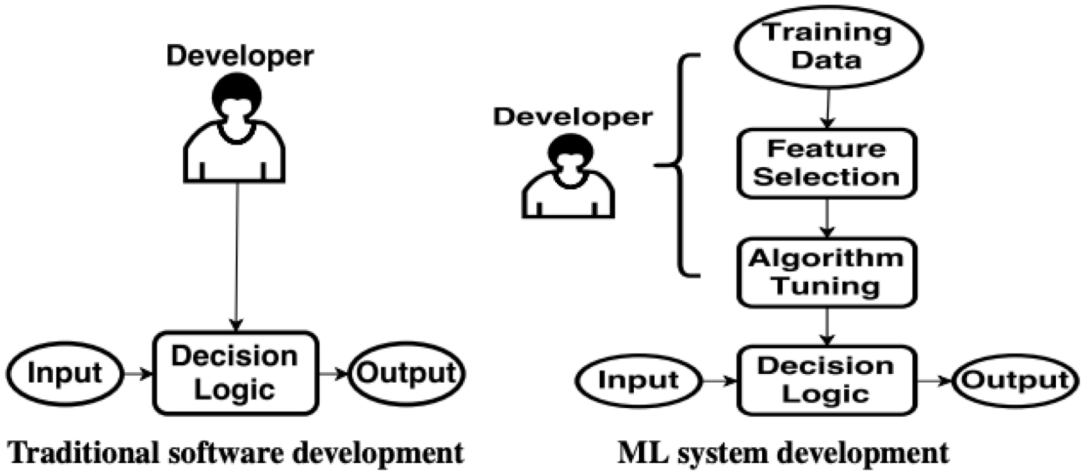
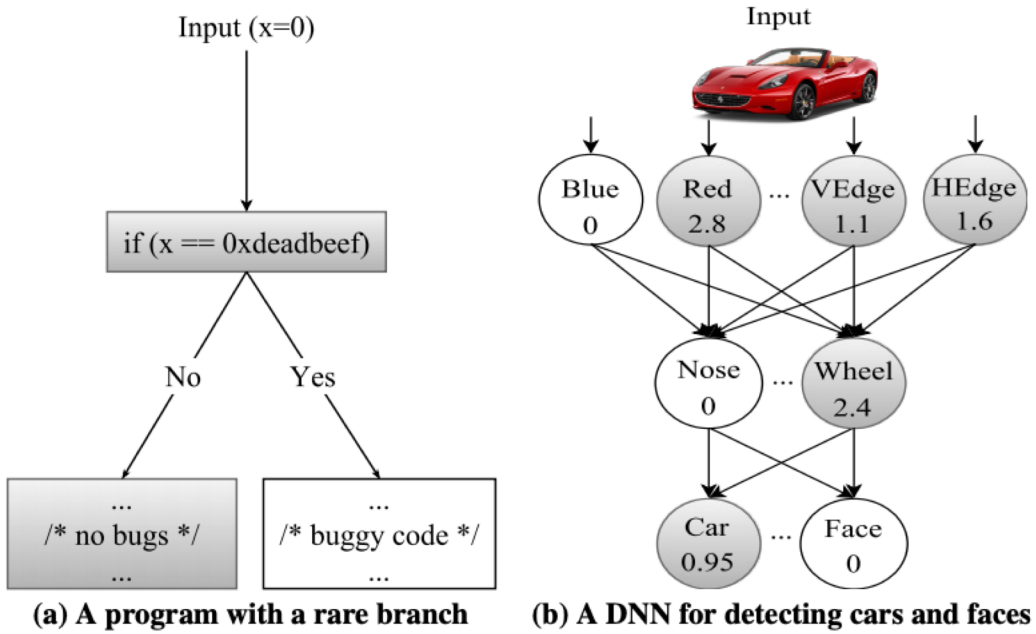


Figure 2: Comparison between traditional and ML system development processes.

Traditional	developers directly specify system logic
DL system	DNNs learn the rules developers indirectly influence (modifying data, feature, architecture)

-> Code coverage is not enough

Traditional vs DL system system



(a) A program with a rare branch (b) A DNN for detecting cars and faces

Figure 4: 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.

Similarity

- (statement/node) perform certain operation
- Output of former (statement/node) are used as a input of next (statement/node)

Coverage

Traditional	rare input: x = 0xdeadbeef -> execute buggy code
DL system	rare input: (nose, red) -> car? face?

Neuron coverage

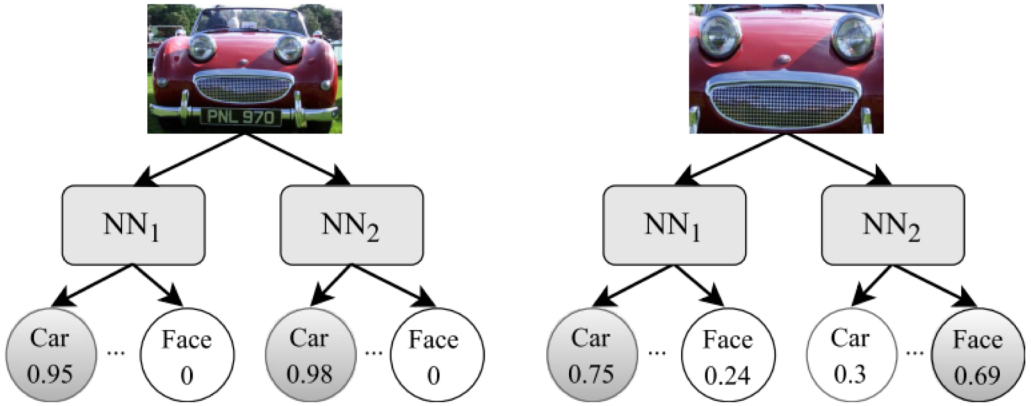
- ratio of unique activated neurons for all test inputs and the total neurons in the DNN

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

N = {n1,n2, ...}	all neurons of DNN
T = {**x1**, **x2**, ...}	all test inputs
out(n, **x**)	function (input: **x**, neuron: n -> output value)
t	threshold of activation

- Objective: generate input that maximize neuron coverage

Cross-referencing



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

Figure 6: Inputs inducing different behaviors in two similar DNNs.

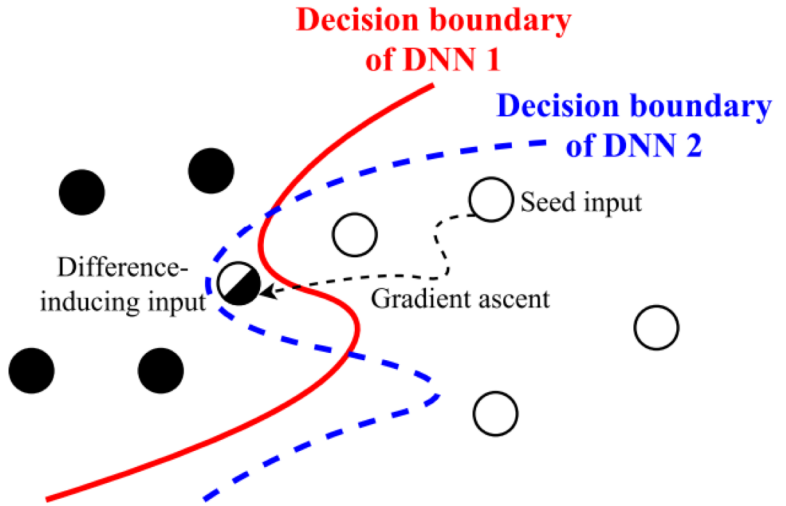


Figure 7: Gradient ascent starting from a seed input and gradually finding the difference-inducing test inputs.

- Multiple DNNs perform similar tasks (trained differently)
- Start from seed input (same output),
  find decision boundaries of DNNs (different output)
- Objective: generate input that maximize differential behavior

Objective function

- Objective1: maximize differential behavior

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

$F_k(\mathbf{x})[c]$: k^{th} DNN predict $\mathbf{x}$ to c

- Objective2: maximize neuron coverage

$$obj_2(\boldsymbol{x}) = f_n(\boldsymbol{x})$$

$f_n(\mathbf{x})$: neuron n's output of input $\mathbf{x}$
$f_n(\mathbf{x})$ > threshold -> neuron activated

- Joint optimization: maximize combination of obj1 and obj2

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

Optimization

- Domain-specific constraint
  : make input valid
  (ex. Image **x**'s pixel value should be in (0, 255))

- Gradient ascent
  : similar with gradient descent (backpropagation)

	objective	input	weight
Gradient ascent	Learn input	variable	constant
Gradient decent	Learn weight	constant	variable

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + s \cdot \boldsymbol{grad}$$

Hyperparameters

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

	role	high	low
λ1	balance output differences of DNNs	lower one DNN's prediction	maintain others' prediction
λ2	balance coverage and differential behavior	focus on coverage	focus on difference-inducing
s	step size in gradient ascent	oscillation	slow
t	threshold for determining if a neuron is activated	hard	easy

Hyperparameters

$obj_2(\boldsymbol{x}) = f_n(\boldsymbol{x})$ such that $f_n(\mathbf{x}) > t$

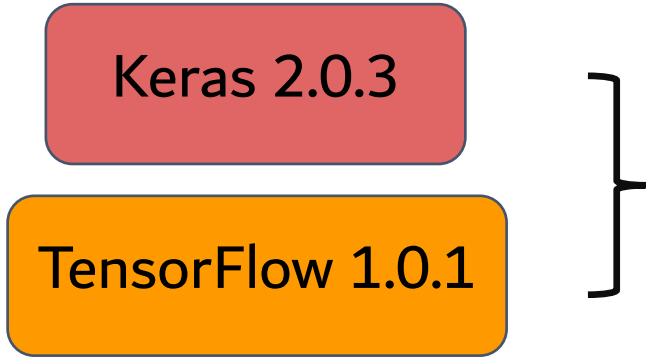
	role	high	low
λ1	balance output differences of DNNs	lower one DNN's prediction	maintain others' prediction
λ2	balance coverage and differential behavior	focus on coverage	focus on difference-inducing
s	step size in gradient ascent	oscillation	slow
t	threshold for determining if a neuron is activated	hard	easy

Hyperparameters

$$x_{i+1} = x_i + s \cdot grad$$

	role	high	low
λ1	balance output differences of DNNs	lower one DNN's prediction	maintain others' prediction
λ2	balance coverage and differential behavior	focus on coverage	focus on difference-inducing
s	step size in gradient ascent	oscillation	slow
t	threshold for determining if a neuron is activated	hard	easy

Implementation & Environment



Efficient gradient computation & support for intercepting outputs of intermediate neurons for calculating neuron coverage

Code: https://github.com/peikexin9/deepxplore

OS: Ubuntu 16.04
CPU: intel i7-6700HQ 2.60Hz (4 cores)
memory: 16GB
GPU: NVIDIA GTX 1070

Dataset & DNNs

Table 1: 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%

Domain-specific constraint (image)

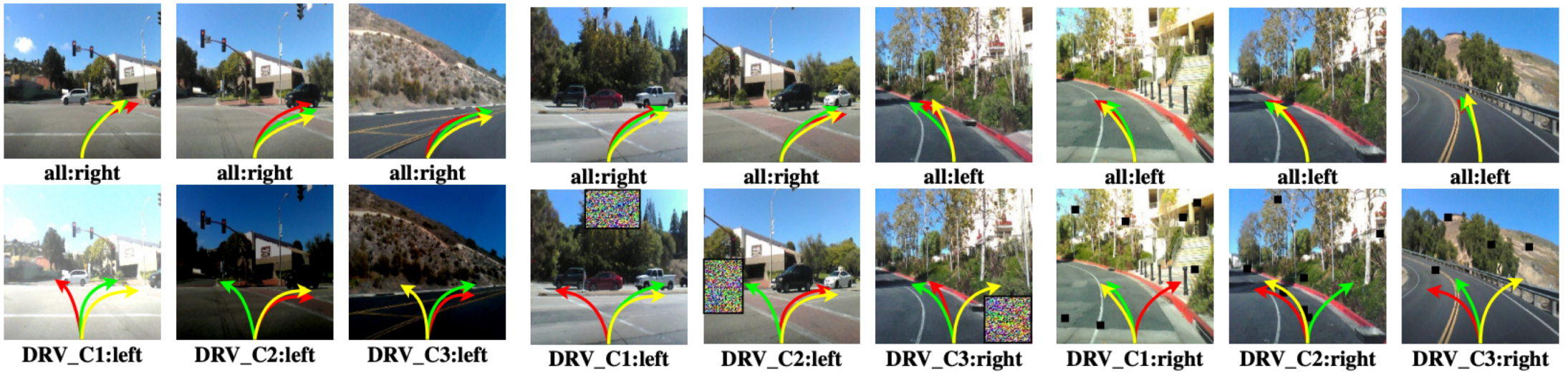
MNIST, ImageNet, Driving

1. lighting effects
   : different intensities of lights

2. occlusion by single small rectangle
   : blocking some parts of a camera

3. occlusion by multiple tiny black rectangles
   : effects of dirt on camera lens

Sample generated input (image)



lighting effects

occlusion by single small rectan

occlusion by multiple tiny rectangles

Domain-specific constraint (Drebin)

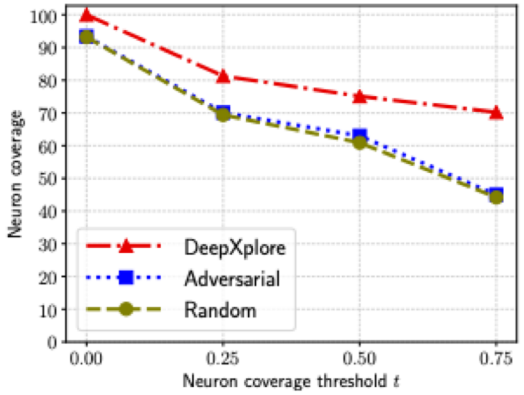
- only allow modifying android manifest file
  : ensure application code is not affected

- only allow adding features
  : avoid insufficient permission

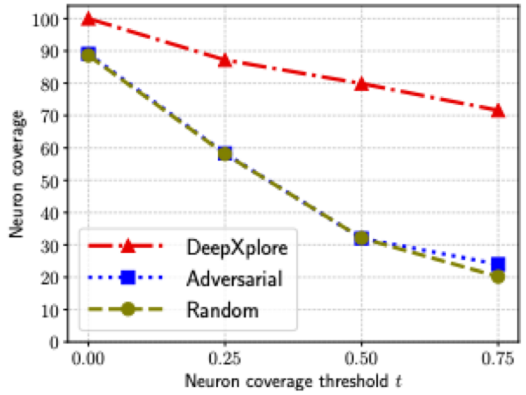
Sample generated input (Drebin)

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

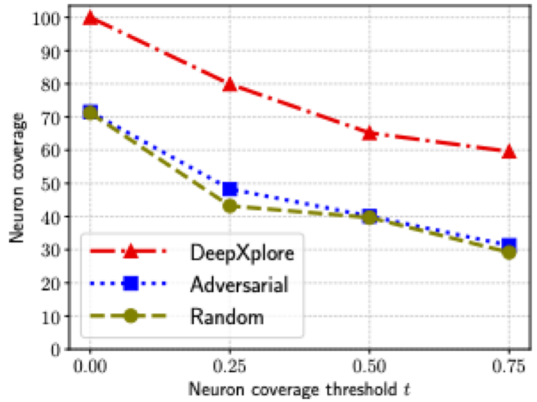
Performance – neuron coverage



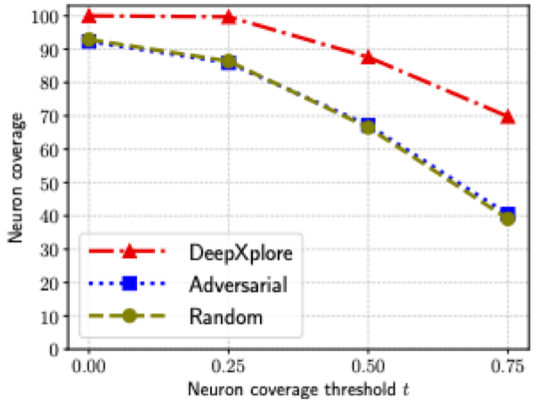
(a) MNIST



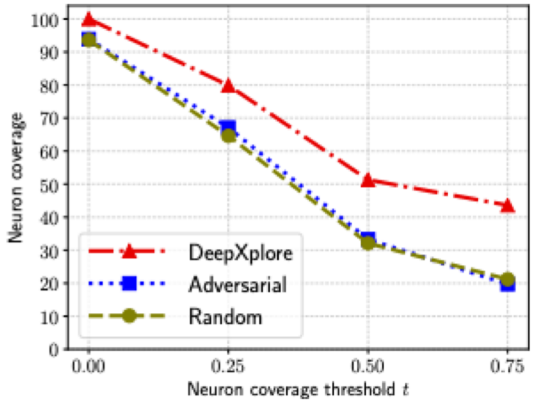
(b) ImageNet



(c) Driving



(d) VirusTotal



(e) Drebin

1. DeepXplore covers
   34.4% (random testing)
   33.2% (adversarial testing)
   more neurons
2. threshold t increases
   -> cover fewer neurons

Performance – execution time

Table 8: Total time taken by DeepXplore to achieve 100% neuron coverage for different DNNs averaged over 10 runs. The last column shows the number of seed inputs.

	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

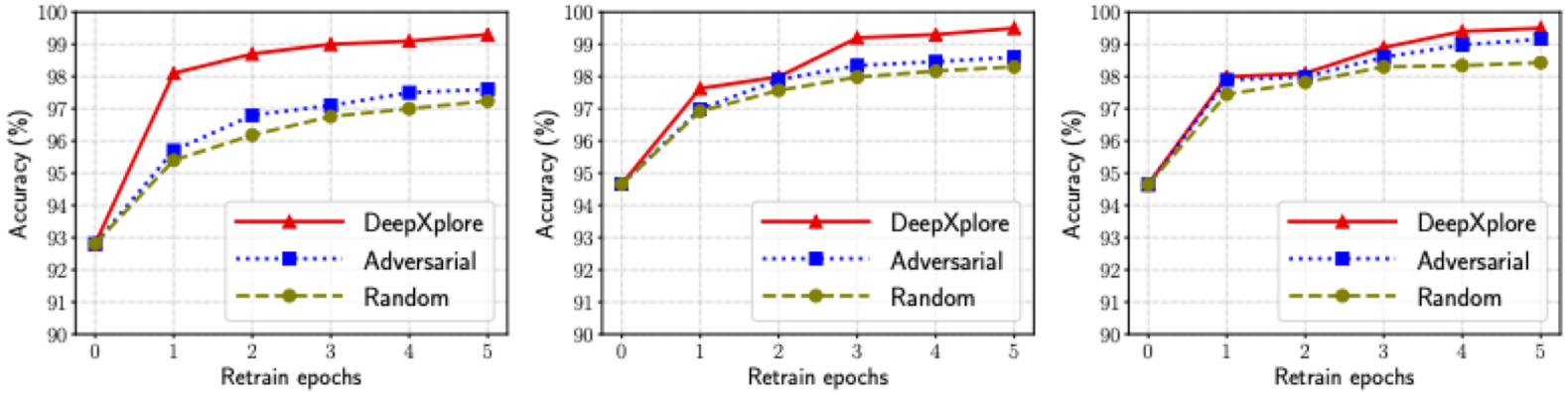
Performance

Table 2: 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.

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

Application

Augmenting training data to improve accuracy



(a) LeNet-1 (b) LeNet-4 (c) LeNet-5

- add 100 new error-inducing samples and retrain the 5 epochs
  : DeepXplore achieve 1~3% more accuracy improvement

Application

Detecting training data pollution attack

- generate error-inducing inputs that are classified differently by unpolluted and polluted versions
- Search sample that are closest to generated inputin terms of structural similarity

30% of the images labeled as digit 9 are mislabeled as 1
-> correctly identify 95.6% of the polluted samples

Summary

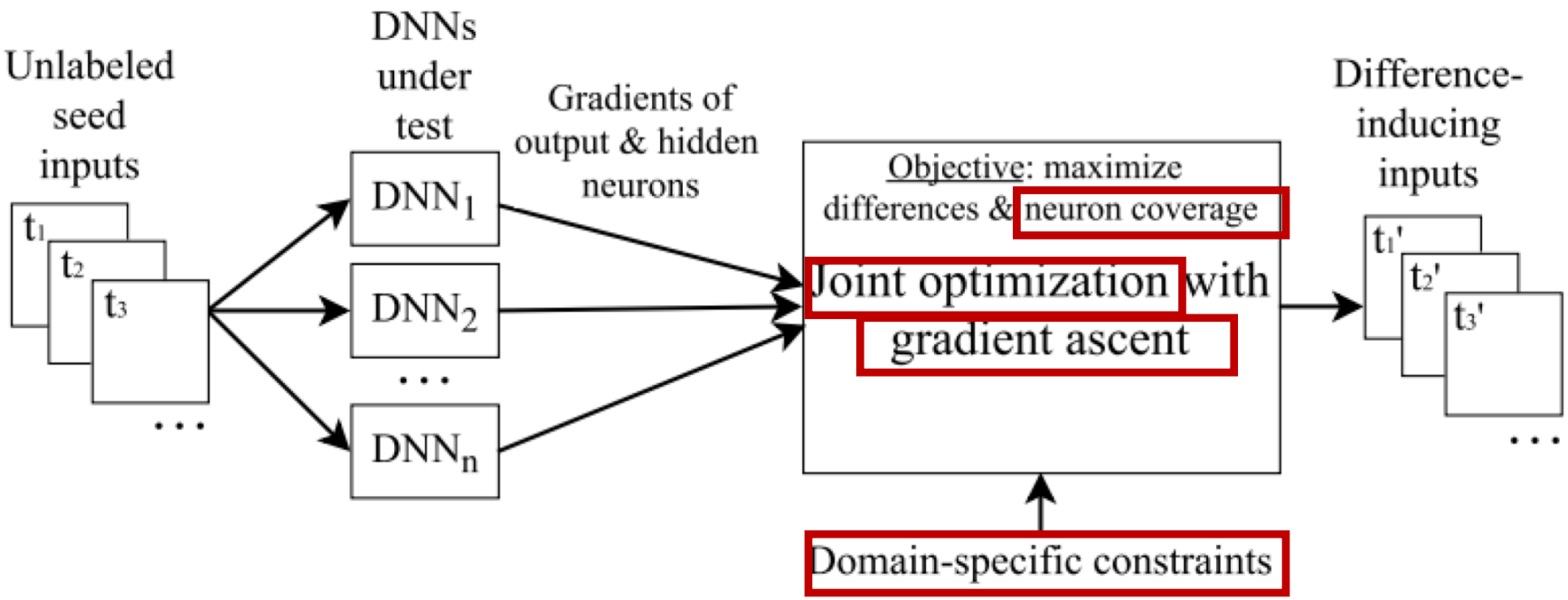


Figure 5: DeepXplore workflow.

exposed thousands of unique incorrect corner case behaviors
in 15 state-of-the-art DL models using five real-world datasets

Question?