A Summary of the paper ‘DeepXplore: Automated Whitebox Testing of Deep Learning Systems’

Topics to discuss

- Understanding keywords
- Focused problem
- Current approaches to solve the problem
- Limitations in current approaches
- Challenges to solve the problem
- Objective functions and terms of algorithm
- Steps of algorithm
- Our results by implementation
- Claimed contributions of the paper
- Limitations of the paper

Understanding keywords

- **Whitebox testing**: A software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs.
  - ➢This method is named so because the software program, in the eyes of the tester, is like a white/transparent box; inside which one clearly sees.

- **Corner Case:** Involves a problem or situation that occurs only outside of normal operating parameters.

Understanding keywords

- **Differential testing**: A popular testing technique that attempts to detect bugs, by providing the same input to a series of similar applications and observing differences in their execution.
  - ➢If n other models all make one prediction for a given input, and one model makes a different prediction, then that model has a bug**.** Differential testing requires at least two DNNs with the same functionality.
- **Neuron coverage:** A metric that looks at the percentage of neurons which are activated during execution of a test suite. Neuron coverage of a set of test inputs is the ratio of the number of unique activated neurons for all test inputs and the total number of neurons in the DNN.
  - ➢We consider a neuron to be activated if its output is higher than a threshold value(0)

Focused problem

- Dependency of existing DL systems manually labeled data to improve accuracy and thus failure in predicting corner case behaviors
  - DL systems are biasedly trained
  - Decision boundaries are different for different DL systems

Current approaches & limitations

- Gather and manually label as much data as possible
  - Time consuming & covers only a small portion of data
- Produce simulated synthetic data
  - Does not consider internals of targeted DL systems
- Adversarial DL systems
  - Must make tiny changes & requires manual checking

Challenges of solving problem

- How to generate inputs that uncover different parts of logics of DL systems
- How to identify erroneous behavior without manual checking

DeepXplore Algorithm

- **Objectives**
  1. Maximizing differential behaviors: The first objective of the optimization problem is to generate test inputs that can induce different behaviors in the tested DNNs, i.e., different DNNs will classify the same input into different classes.
  2. Maximizing neuron coverage: The second objective is to generate inputs that maximize neuron coverage. This can be done by iteratively picking inactivated neurons and modifying the input such that output of that neuron goes above the neuron activation threshold.

  Joint optimization: The above-mentioned objectives have been combined as a joint optimization problem and maximized based on gradient ascent algorithm.

Functions and terms

First objective function:

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

Joint objective function:

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

- **λ1** balances the objectives between minimizing one DNN's prediction for a certain label and maximizing the rest of DNNs' predictions for the same label. Larger λ1 puts higher priority on lowering the prediction value/confidence of a particular DNN while smaller λ1 puts more weight on maintaining the other DNNs' predictions.
- **λ2** provides balance between finding differential behaviors and neuron coverage. Larger λ2 focuses more on covering different neurons while smaller λ2 generates more difference-inducing test inputs.

DeepXplore Algorithm Implementation

- Takes unlabeled inputs as the seeds
- Checks if the all the models produce same predictions by current image
- If no, then
  - Update neuron coverage and calculate neuron covered
  - Save the image as INPUT ALREADY CAUSING DIFFERENCE
- If yes, then
  - Find a random inactivated neuron from each models
  - Select a model & calculate objective function for the model being tested
  - Calculate gradient of the objective function with respect to input image
  - Apply domain specific constraints on image using gradient value
  - Check if predictions by all models match
    - If not, then update neuron coverage and calculate neuron covered for each model and save image as Generated new image that is a corner case for current model .
    - If yes, then again calculate gradient and apply domain constraints until predictions mismatch and iterations not finished

Some other required terms

- Step(s) controls the step size used during iterative gradient ascent.
  - Larger s may lead to oscillation around the local optimum
  - Smaller s may need more iterations to reach the objective.
- Threshold(t) is the threshold to determine whether each individual neuron is activated or not.
  - Finding inputs that activate a neuron become increasingly harder as t increases.

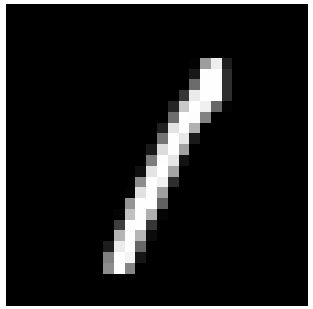
Implementation:

- Dataset: MNIST
  - Unlabeled inputs as seeds are taken from test data containing 10000 images
- Models: LeNet-1, LeNet-4 & LeNet-5
- Domain constraints used:
  - Light: restricts image modifications so that DeepXplore can only make the image darker or brighter without changing its content.
  - Occlusion: constraint simulates the effect of the camera lens that may be accidentally or deliberately occluded by a single small rectangle
  - Blackout: constraint simulates the effect of small dirt in the camera lens

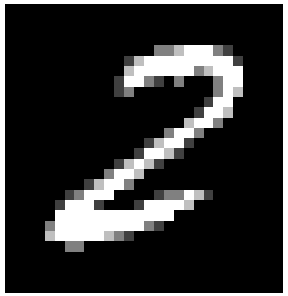
Implementation verification:

Different lighting conditions:

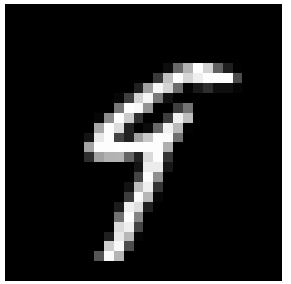
ALL: 1



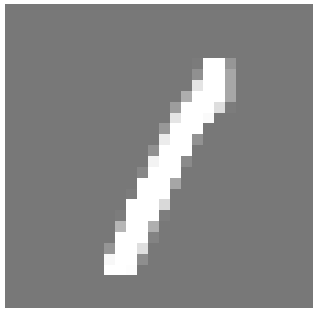
ALL: 2



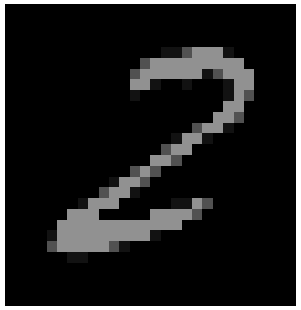
ALL: 9



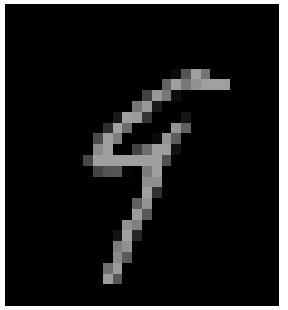
Model 1: 8



Model 2: 3



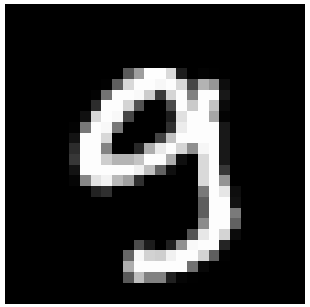
Model 3: 5



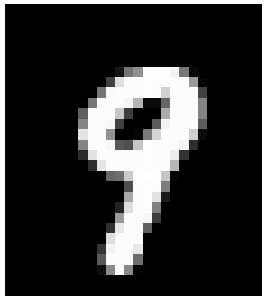
Implementation verification:

Occlusion with a small single rectangle:

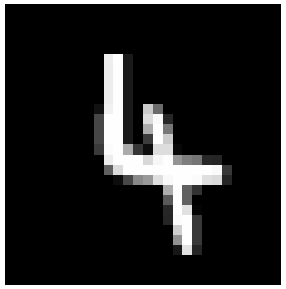
ALL: 9



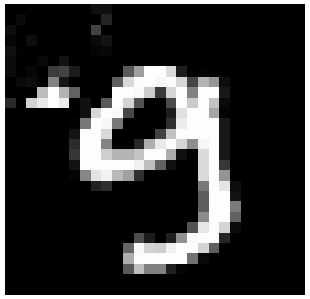
ALL: 9



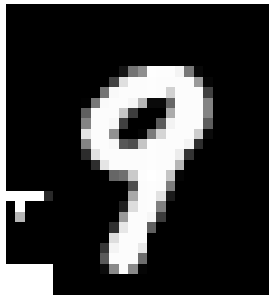
ALL: 4



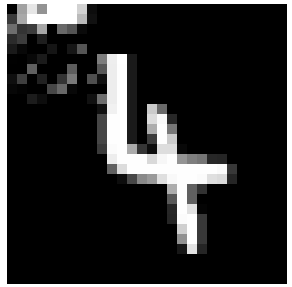
Model 1: 3



Model 2: 3



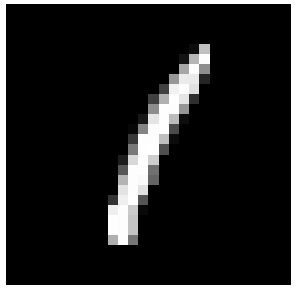
Model 3: 8



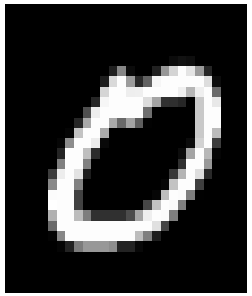
Implementation verification:

Blackout:

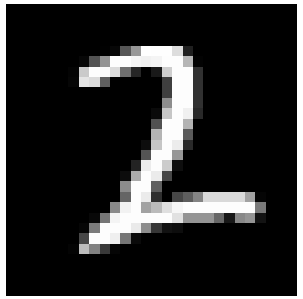
ALL: 1



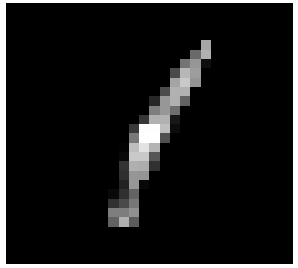
ALL: 0



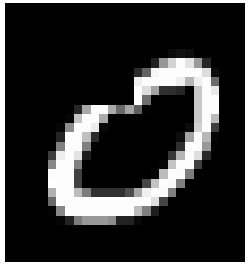
ALL: 2



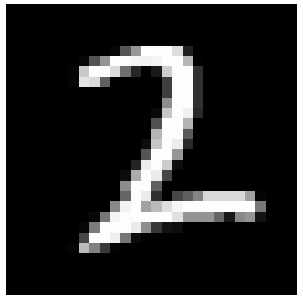
Model 1: 4



Model 2: 7



Model 3: 1



Claimed contributions of the paper:

- Solving joint optimization problem
- Supports adding custom constraints
- Generating images of corner case behaviors
- Retraining DNN models using generated images and improvement of accuracy by 1-3%

Limitations of the paper:

- In case of slightly different DNNs, DeepXplore will take longer to find difference-inducing inputs
- If all the tested DNNs make the same mistake, DeepXplore cannot generate the corresponding test case