# Fuzz Testing based Data Augmentation to Improve Robustness of Deep Neural Networks

Xiang Gao*
National University of Singapore, Singapore
gaoxiang@comp.nus.edu.sg

Ripon K. Saha
Fujitsu Laboratories of America, Inc, USA
rsaha@us.fujitsu.com

Mukul R. Prasad
Fujitsu Laboratories of America, Inc, USA
mukul@us.fujitsu.com

Abhik Roychoudhury
National University of Singapore, Singapore
abhik@comp.nus.edu.sg

## ABSTRACT

Deep neural networks (DNN) have been shown to be notoriously brittle to small perturbations in their input data. This problem is analogous to the over-fitting problem in test-based program synthesis and automatic program repair, which is a consequence of the incomplete specification, i.e., the limited tests or training examples, that the program synthesis or repair algorithm has to learn from. Recently, test generation techniques have been successfully employed to augment existing specifications of intended program behavior, to improve the generalizability of program synthesis and repair. Inspired by these approaches, in this paper, we propose a technique that re-purposes software testing methods, specifically mutation-based fuzzing, to augment the training data of DNNs, with the objective of enhancing their robustness. Our technique casts the DNN data augmentation problem as an optimization problem. It uses genetic search to generate the most suitable variant of an input data to use for training the DNN, while simultaneously identifying opportunities to accelerate training by skipping augmentation in many instances. We instantiate this technique in two tools, Sensei and Sensei-SA, and evaluate them on 15 DNN models spanning 5 popular image data-sets. Our evaluation shows that Sensei can improve the robust accuracy of the DNN, compared to the state of the art, on *each* of the 15 models, by upto 11.9% and 5.5% on average. Further, Sensei-SA can reduce the average DNN training time by 25%, while still improving robust accuracy.

## CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Software and its engineering** → **Search-based software engineering**; **Software testing and debugging**.

## KEYWORDS

Genetic Algorithm, DNN, Robustness, Data Augmentation

*This work was mainly done when the author was an intern at Fujitsu Labs of America.


*ICSE '20, May 23–29, 2020, Seoul, Republic of Korea*

ACM ISBN 978-1-4503-7121-6/20/05…$15.00
https://doi.org/10.1145/3377811.3380415

**ACM Reference Format:**
Xiang Gao, Ripon K. Saha, Mukul R. Prasad, and Abhik Roychoudhury. 2020. Fuzz Testing based Data Augmentation to Improve Robustness of Deep Neural Networks. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3377811.3380415

## 1 INTRODUCTION

Programming by examples (PBE) such as test-based program synthesis and automated program repair [1, 18], automatically generates programs that conform to the specification indicated by the given examples. Since the given examples usually constitute an incomplete specification of intended behavior, the generated program may over-fit the given examples and thereby not exhibit intended behaviors on the un-seen part of the input space. Over-fitting is a common problem in test-based program synthesis [1, 9] and automated program repair [18, 25]. Similarly, machine/deep learning [7] also faces the over-fitting problem. In machine learning, a model is usually learned from a training set (of inputs), and then deployed on a testing set. Over-fitting in machine learning systems impacts two related but distinct properties of such systems: (1) inadequate *standard generalization*, where the trained model shows high accuracy on the training set, but cannot be generalized to data points outside the training set, e.g., the testing set, and (2) inadequate *robust generalization*, where the model show high accuracy on both training and testing sets, but cannot be generalized to inputs that are small perturbations of training/testing inputs; these small perturbations may still constitute legal inputs, but the learned model often mis-classifies such inputs. Robust generalization renders the learned models resilient against such small perturbations.

Standard generalization does not imply robustness generalization. For instance, a model, even with 99% accuracy on its testing dataset, could misclassify the input variations generated by simple spatial transformation (e.g., rotation) with high confidence [8]. Figure 1 presents an example from the GTSRB dataset. Robust generalization itself can be of two types, depending on the operation context. In a security setting, robust generalization against adversarial inputs aims to protect a model against a powerful adversary, that can modify the input in sophisticated and targeted ways (i.e., an *attack*), such as obscuring a few specific pixels in an image, to make the model mis-predict. This scenario has been popularized by the work on adversarial testing [11, 12, 23, 29] and more recently, adversarial robust generalization [4, 14, 17, 21, 31, 45]. A second scenario, however, is perturbations due to natural variations in


(a) Original image is correctly classified (b) Incorrect classification after one degree rotation

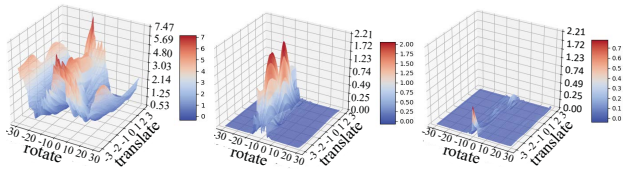
**Figure 1: A motivating example from the GTSRB dataset**

environmental conditions when capturing input data. Examples of this are variations due to weather conditions (e.g., snow, fog) or the illumination (e.g., day or night) for the *same location*, for a self-driving car, or variations in the position, angle, or exposure settings for a camera capturing the same subject. Recent work has highlighted the poor robustness of Deep Learning models to such variations [19, 24, 30, 47]. In this work, we focus on the latter scenario and investigate how software testing techniques can be applied for the robust generalization of Deep Neural Networks (DNNs) to natural environmental variations. Although we present our approach in the context of DNNs, the core ideas are more broadly applicable to learning systems.

In the field of test driven program synthesis and program repair, a prominent class of recent techniques improves the generalization of generated programs by augmenting existing test suites [9, 40, 41, 43]. Test case generation techniques (such as random testing, search-based evolutionary testing, symbolic execution, grey-box fuzzing) have been used to augment existing specifications of intended program behavior [40, 41, 43]. In particular, coverage-based grey-box fuzzing approaches, such as AFL [32], have shown utility in augmenting existing test suites for program repair [10]. In grey-box fuzzing algorithms, i[1]nputs are randomly mutated to generate new inputs and higher priority is assigned to inputs that exercise new and interesting paths. The intuition behind these techniques is that covering more program paths enables us to find representative tests and covers more program functionality.

At a conceptual level, training AI models is analogous to program synthesis [1]. A learning system generates a model that can clearly classify a given input to its corresponding label. Specifically, neural network models can be considered as layers of program statements connected by a parameter matrix. Given a set of training data (inputs) with labels (outputs), the knowledge acquired during training is encoded into parameters that connect layers. Thus, it is natural to ask if software test generation based augmentation techniques, which have been successfully applied to improve generalization of program synthesis and repair, can be re-purposed for robust generalization of DNNs. This paper develops this idea, by employing mutation-based fuzzing for data augmentation of DNNs.

It is a common practice to boost the standard generalization of DNNs, using basic data augmentation, where, *during the initial training*, each training input is substituted by a (single) randomly-generated, label-preserving variant [16, 27]. A more sophisticated version of this is done in *mixup* [46], a recently proposed state-of-the-art data augmentation technique, that trains a DNN on convex combinations of pairs of data and their labels. However, as shown in Section 5, while this boosts the standard generalization and the robust generalization to adversarial examples, it has limited impact on the robust generalization to *natural variations*. Further, the space of potential label-preserving transformations and their parameter




(a) Standard model (b) State-of-the-art (c) Sensei

**Figure 2: The loss for various transformations of the motivating example after augmented training.**

values is very rich [8, 24, 30]. Thus, naïve augmentation with all or several variants is not viable either — it would significantly increase the size of the training set and training time. In the past, robust optimization [2], based on gradient descent, has been used to defend DNNs against adversarial examples [8, 21]; these techniques try to generate the worst variant, based on a loss function, and add it to the training set. However, as shown in Figure 2(a), the input space of spatial transformations, which are prominently represented in natural environmental variations (for vision applications), is highly non-convex. The gradient descent techniques perform rather poorly in such scenarios [8], and therefore not applicable for our problem. Even a state-of-the-art augmentation approach [8] performs poorly for such non-convex space as shown in Figure 2(b). It is important to note that while our data augmentation technique generates and uses the generated data *during initial training*, almost all techniques to improve robustness [14, 15, 17, 24, 30, 31] generate adversarial examples by analyzing a trained model and subsequently *re-train* the model on this new data. Our evaluation (Section 5) demonstrates that our technique provides better robust generalization than the latter approach. Concurrent with our work, Yang et al. have proposed an orthogonal approach to improving DNN robustness to spatial variations [42]. Their approach modifies the loss function of the DNN by adding an invariant-inducing regularization term to the standard empirical loss. This is complementary to our proposed data augmentation based mechanism of improving robustness. Exploring the combination of these two approaches could present an interesting opportunity for future work.

*Proposed technique.* In this paper, we propose a new algorithm that uses guided test generation techniques to address the data augmentation problem for robust generalization of DNNs under natural environmental variations. Specifically, we cast data augmentation problem as an optimization problem, and use genetic search on a space of the natural environmental variants of each training input data, to identify the *worst* variant for augmentation. The iterative nature of the genetic algorithm (GA) is naturally overlaid on the multi-epoch training schedule of the DNN, where in each iteration, for each training data, the GA explores a small population of variants and selects the worst, for augmentation, and further uses it as the seed for the search in the next epoch, gradually approaching the worst variant, without explicitly evaluating all possible variants. Further, we propose a novel heuristic technique, called *selective augmentation* which allows skipping augmentation completely for a training data point in certain epochs based on an analysis of the DNN's current robustness around that point. This allows a substantial reduction in the DNN's training time under augmentation. The contributions of this paper include:

- We formalize the data augmentation for robust generalization of DNNs, under natural environmental variations, as a search problem and solve the search problem using fuzz testing approaches, specifically using genetic search.
- To reduce the overhead caused by data augmentation, we propose a selective data augmentation strategy, where only part of data points are selected to be augmented.
- As a practical realization of the proposed technique, we implement two prototype tools Sensei and Sensei-SA.
- We evaluate the proposed approach on 15 DNN models, spanning 5 popular image data-sets. The results show that the Sensei can improve the robust accuracy of *all* the models, compared to the state of the art, by upto 11.9% and 5.5% on average. Sensei-SA can reduce DNN average training time by 25%, while still improving robust accuracy. Currently, our approach has only been evaluated on image classification datasets. However, conceptually, it may have wider applicability.

# 2 BACKGROUND

## 2.1 Fuzz Testing

Fuzz testing is a common and practical approach to find software bugs or vulnerabilities, where new tests are generated by mutating existing seeds (inputs). By selecting the seeds to mutate and controlling the number of generated mutations, we can effectively and efficiently achieve a certain testing goal (e.g. high code coverage). Algorithm 1 briefly describes how greybox fuzzing (e.g. AFL [32]) works. Given a set of initial seed inputs $S$, the fuzzer chooses $s$ from $S$ in a continuous loop. For each $s$, the fuzzer determines the number of tests, which is called the *energy* of $s$, to be generated by mutating $s$. Then, we execute program $P$ with the newly generated test $s'$ (line 6) and monitor the run-time behavior. Whether $s'$ is added to the seed queue is determined by a fitness function (line 7), which defines how good test $s'$ is to achieve a certain testing goal.

**Algorithm 1:** Test generation via greybox fuzzing

```
   Input: seed inputs S; program P
1  while timeout is not reached do
2      s := chooseNext(S);
3      energy := assignEnergy(s);
4      for i from 1 to energy do
5          s' := mutate(s);
6          execute (P, s');
7          if fitness(s') > threshold then
8              S := S ∪ s';
9      end
10 end
```

## 2.2 Training Deep Neural Networks

Given a DNN model $M$ with a set of parameters (or *weights*) $\theta \in R^p$ being trained on a training dataset $D$ that consists of pairs of examples $x \in R^d$ (drawn from a representative distribution) and corresponding labels $y \in [k]$, the objective of training $M$ is to infer optimal values of $\theta$ such that the aggregated loss over $D$ computed by $M$ is minimum. Following the treatment in [21], this can be expressed as the following minimization problem:

$$\min_{\theta} E_{(x,y)\sim D}[L(\theta, x, y)] \tag{1}$$

where $L(\theta, x, y)$ is a suitable cross-entropy loss function for $M$ and $E_{(x,y)\sim D}()$ is a risk function that (inversely) measures the accuracy of $M$ over its training population. In practice, the solution to this problem is approximated in a series of iterative refinements to the values of $\theta$, called *epochs*. In each epoch, $\theta$ is updated with the objective of minimizing the loss of training data.

## 2.3 Robustness of DNNs

DNNs have demonstrated impressive success in a wide range of applications [3, 48]. However, DNNs have also been shown to be quite brittle, i.e., not *robust*, to small changes in their input. Specifically, a DNN $M$ may correctly classify an input $x$ with its corresponding label $l$, but incorrectly classify an input $x + \delta$ that is *similar* to $x$, with label $l'$, where $l \neq l'$. Although our ideas are broadly applicable, the sequel assumes a DNN performing an image classification task. In this context, $x$ is an image, and $x + \delta$ a perceptually similar (to the human user) variant of $x$.

As discussed earlier, this work targets robustness issues arising from natural, environmental perturbations $\delta$ in the input data and *not* perturbations $\delta$ constructed adversarially, in a security context. The allowed perturbations $\delta$ can be represented as a neighborhood $S$ around input $x$, such that $\forall \delta \in S, x + \delta$ constitutes legal input for $M$ and is perceptually similar to $x$ and hence carries the same label $l$. $S$ can be simulated through a set of parameterized transformations $T(\vec{\rho}, x) = \{t_1(\rho_1, x), t_2(\rho_2, x), \ldots, t_k(\rho_k, x)\}$ (where $\vec{\rho} = \langle \rho_1, \rho_2, \ldots, \rho_k \rangle$), including common image transformations such as rotation, translation, brightness or contrast changes, *etc.*, as done by recent work on robustness testing of DNNs [8, 24, 30]. Alternatively, $S$ can be synthesized using generative models such Generative Adversarial Neural Networks (GANs) [20, 47]. We employ the former approach. Specifically, a variant $x'$ of image $x$ can be computed by applying the composition of transformations $t_1, t_2, \ldots, t_k$ in sequence (denoted by $t$) on $x$, as:

$$x' = t(\vec{\rho}, x) = t_k(\rho_k, \ldots, t_2(\rho_2, t_1(\rho_1, x)) \ldots) \tag{2}$$

## 2.4 Data Augmentation for DNNs

Since DNNs self-learn the relevant features from the training data they may learn irrelevant features of the specific data (i.e., overfitting) and generalize poorly to other data [16]. To improve (standard) generalization of DNNs it is common practice to perform a basic form of data augmentation where, during training, in each epoch, each training data is replaced by a variant created by randomly applying some sources of variation or noise (for example the transformations $T$ above). As shown in Section 5, this basic strategy also boosts robust generalization but with significant room for improvement. Data augmentation can be performed in mainly two ways from the training perspective: i) **during initial training:** synthetic data is generated on-the-fly based on some heuristics and then augmented with the training data during the training of the original model, ii) **retraining:** in a two-staged fashion where in the first step, additional data are selected based on the feedback on the original model, and then in the second step, the model is retrained with the augmented data.

# 3 SENSEI: AN AUTOMATIC DATA AUGMENTATION FRAMEWORK FOR DNNS

Sensei targets improving the robust generalization of a DNN *in-training*, under natural environmental variations, by effectively augmenting the training data. In general, data augmentation could involve adding, removing, or replacing an arbitrary number of training data inputs. However, Sensei, like several augmentation approaches [8, 21], implements a strategy of either replacing each data with a suitable variant or leaving it unchanged. Thus, the total size of the training dataset is also unchanged. Thus, the key contribution of Sensei is to identify the optimal replacement for each training data. In addition, we introduce an optimized version of Sensei called Sensei-SA to optimize the training time by potentially skipping augmentation for some data inputs.

## 3.1 Problem Formulation

The task of training a DNN under robust generalization can be cast as modified version of Equation 1, where, in addition to optimizing for parameters $\theta$ we also need to select, for each training data input $x$, a suitable variant $x' = x + \delta$, where $\delta \in S$. Following [21] this can be cast as the following saddle-point optimization problem:

$$\min_{\theta} E_{(x,y)\sim D}[\max_{\delta \in S} L(\theta, x+\delta, y)] \quad (3)$$

Sensei approximates the solution of this optimization problem by decoupling the inner maximization problem (which solves for $\delta$) from the outer minimization problem (which optimizes $\theta$). This is done by allowing the usual iterative epoch-based training schedule to optimize for, but in each epoch, for each training data $x$, solving the inner maximization problem to find the optimal variant $x+\delta$. Specifically, given the set of transformations $T(\vec{\rho}, x)$ defining neighborhood $S$ and using an overloaded definition of $S$ in terms of the parameter vector $\vec{\rho}$, Sensei solves following optimization problem:

Definition 1 (*Augmentation target*). *Given a seed training data input $x$ and transformation function $t(\vec{\rho}, x)$ defining neighborhood $S$ of $x$, find $\vec{\rho}$ yielding the optimal variant $x'$ (per Equation 2) to optimize:*

$$\max_{\vec{\rho} \in S} L(\theta, t(\vec{\rho}, x), y) \quad (4)$$

## 3.2 An Overview

In order to solve the optimization problem defined in Equation 4 effectively and efficiently, our proposed approach includes two novel insights. Our first insight is that although traditional data augmentation techniques improve the robust generalization by training the DNN with some random variations of the data-points, a fuzz testing based approach such as guided search may be more effective to find optimal variants of data points to train the DNN, and hence, to improve the robust generalization. Our second insight is that not all data points in the training dataset are difficult to learn. Some data points represent ideal examples in the training set while some are confusing. Therefore, treating all the points similarly regardless of their difficulties levels may result in waste of valuable training time. We may save a significant amount of training time by spending the augmentation effort on only the challenging data-points while skipping augmenting for ideal or near-ideal examples.

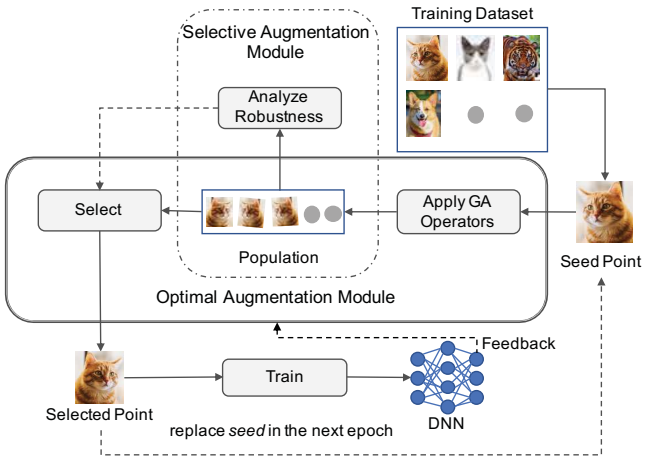



**Figure 3: An overview of Sensei for one seed image in one epoch. Given images are only for illustration purposes without proper scaling.**

**Algorithm 2:** Overall algorithm

**Input:** Training set $(X, Y)$, number of training epochs $nE$, population size *popSize*, crossover probability $p$
**Output:** Model $M$

```
1  epoch := 1;
2  M := train(X, Y);   // train M with original data in first epoch
3  for i in range(0, |X|) do
4      Pop_i := randomInitPopulation(X[i]);
5      isPWRobust_i := False;
6  end
7  while epoch < nE do
8      epoch := epoch + 1;
9      for i in range(0, |X|) do
10         if isPWRobust_i then
11             isPWRobust_i := isRobust(X[i]);
12             continue;                          // selective augmentation
13         children := genPop(Pop_i, p, popSize);          // Alg.3
14         f := fitness(M, children);                      // Equation 5
           // replace original data with child with highest fitness
15         X[i] := selectBest(children, f);
16         Pop_i := select(children, f);                   // new population
17         isPWRobust_i := pointWiseRobust(X[i], Pop_i);
18     end
19     M := train(X, Y);
20 end
```

Figure 3 presents an overview of the proposed framework, Sensei, for one seed image and for one epoch. There are two main components in Sensei: i) optimal augmentation and ii) selective augmentation, which basically realize the two aforementioned insights. Algorithm 2 provides even further detail on how the overall approach is overlaid on the multi-epoch training schedule of $M$. Sensei starts training $M$ with the original data point in the first epoch (line 2). However, from the second epoch, the *optimal augmentation* module efficiently finds the most potential variation ($x'$)

that challenges $M$ the most, replace $x$ with $x'$ and use $x'$ for training (line 13-16). The *selective augmentation* module is intended to optimize the training time. When it is enabled, Sensei does not augment every data-point ($x$) right away. Rather, the *selective augmentation* module first determines whether the current state of $M$ is robust around $x$. If so, Sensei-SA keeps skipping the augmentation of $x$ (line 10-12, 17) until $M$ becomes unrobust around $x$. Note that, Sensei is *in-training* data augmentation approach, i.e., data generation and augmentation happen on-the-fly during training.

## 3.3 Optimal Augmentation

Theoretically, a DNN could be trained with infinite number of realistic variants ($x'$) to increase the robust generalization. However, it is impractical to explore many variations of original data-points in a brute-force fashion. Therefore, the main challenge in automatic data augmentation is identifying the optimal variations of data-points efficiently that would force the model to learn the correct feature of the representing class. In Sensei, our key insight is that since the genetic algorithm is well-known to explore a large search space efficiently to find optimal solutions by mimicking evolution and natural selection, we can effectively employ it to find an optimal variant for each data-point in each epoch to improve the robust generalization. Furthermore, the iterative nature of genetic algorithms naturally gets overlaid on the multi-epoch training of the DNN, which makes the search very efficient.

Adapting genetic algorithms (GA) to any problem involves design of three main steps: (i) representation of *chromosomes*, (ii) generation of *population* using *genetic operators*, and iii) mathematical formulation of a *fitness function*.

*3.3.1 Representation of Chromosome.* In genetic algorithms(GA), a *chromosome* consists of a set of *genes* that defines a proposed solution that the GA is trying to solve. In Sensei, we represent a chromosome as a set of operations that would be applied on a given input to get the realistic variations, which is basically the transformation vector $\vec{\rho} = \langle \rho_1, \rho_2, \ldots, \rho_k \rangle$ described in Section **??**. For instance, we can derive a realistic variation ($x'$) of image ($x$) by rotating $x$ by one degree and then translating it by one pixel, simulating the angle and movement of camera in real life.

*3.3.2 Generation of population.* In GA, a population is a set of chromosomes that represents a subset of solutions in the current generation. In Sensei, the initial population, which is the population in the first epoch, are created randomly. In subsequent generations (epochs), the population is constituted through two genetic operators: *mutation and crossover* and then through a selection technique.

Given the current population, a crossover probability and population size, Sensei applies mutation and crossover operations on the chromosomes in the current population to generate a new population, as presented in Algorithm 3. Mutation is performed by randomly changing a single operation (change parameter) in the chromosome. Crossover is done to create a new chromosome by merging two randomly selected existing chromosomes. Specifically, given two random chromosomes: $c_1 = \{rotation\text{: } 1, translation\text{: } 2, shear\text{: } -0.15\}$ and $c_2 = \{rotation\text{: } -1, translation\text{: } -3, shear\text{: } 0.1\}$, the *crossover* operator

**Algorithm 3:** Generation of population

**Input:** Current Population *Pop*, crossover probability $p$, population size *popSize*
**Output:** OffSpring *children*

```
1  children := {};
2  while size(children) < popSize do
3      r := U(0, 1);
4      if r<p then                                   // use crossover
5          x1, x2 := selectParents(Pop);
6          x1', x2' := crossover(x1, x2);
7      else                                          // use mutate
8          x := selectParent(Pop);
9          op := randomSelectOp(Operations);
10         x1' := mutate(x, op);
11     end
12     if isValid(x1') then
13         children = children ∪ x1'
14     end
15 end
16 return children;
```

first generates a random number, $r$ between 1 and the chromosome length ($l$) and merges $c_1$ and $c_2$ by taking 1 to $r$ transformations from $c_1$ and $r+1$ to $l$ transformations from $c_2$ to form a new chromosome $c_n$. For the given example, $r = 2$ produces $c_n = \{rotation\text{: } 1, translation\text{: } -3, shear\text{: } 0.1\}$. Sensei applies either mutation or crossover operation based on the given crossover probability. It should be noted that once a new chromosome is generated through the mutation or crossover, it is validated to make sure that it is within the range of each transformation that we set globally (line 12). Furthermore, Sensei always applies the resulting transformation vector (chromosome) on the original image (as opposed to applying on an already transformed data) to prevent the resulting data from being unrealistic. Once the new population is generated, they are evaluated and only best set is passed as a current population for the next generation (line 17 in Algorithm 2). The best set is selected through a fitness function.

*3.3.3 Fitness function.* In GA, a fitness function evaluates how close a proposed solution (chromosome) is compared to an optimal solution. The design of a fitness function plays an important role in GA since if the fitness function becomes the bottleneck of the system, the entire system would be inefficient. Furthermore, the fitness function should be intuitive and clearly defined to measure the quality of a given solution. In Sensei, we define the fitness function based on the empirical loss of the DNN. More specifically, since the training of DNN focuses on minimizing loss across the entire training data-set, the variant that suffers in more loss by the DNN should be used in the augmented training to make the DNN more robust. Formally:

$$f_{loss}(x') = L(\theta, x', y) \tag{5}$$

**Other metrics as fitness function.** Any metric that quantifies the quality of a DNN with respect to a test input may be used as a fitness function. Some of the concrete examples include neuron

**Table 1: Short data-set descriptions and statistics**

| Data-set | #Train | #Test | #CL | #MD | Description |
|---|---|---|---|---|---|
| GTSRB | 38047 | 12632 | 43 | 4 | German Traffic Sign Benchmark |
| FM | 60000 | 10000 | 10 | 3 | Zalando's article |
| CFR | 50000 | 10000 | 10 | 4 | Object recognition |
| SVHN | 73257 | 26032 | 10 | 2 | Digit recognition |
| IMDB | 345693 | 115231 | 5 | 2 | Face data-set with gender & age labels |

CFR: CIFAR-10 #CL: #Classes #MD: # DNN Models

coverage [24] and surprise adequacy [15]. Nevertheless, the computation of the fitness function should be reasonably fast so that it does not become the bottleneck of the system.

### 3.4 Selective Augmentation

Unlike traditional techniques that augment all the data-point in the training set irrespective their nature, Sensei-SA skips data-points that are already classified by $M$ robustly. Therefore, the selective-augmentation technique is solely based on the robustness analysis of $M$ w.r.t. a data-point $x$. We formalize the robustness w.r.t. a data-point as *point-wise robustness* which could be determined based on the following two kinds metrics:

**Classification-based robustness.** A model is *point-wise robust* w.r.t. a data-point $x$ if and only if it classifies $x$ and *all* the label preserving realistic variations ($x'$) correctly.

**Loss-based robustness.** A model is point-wise robust w.r.t. a data-point $x$ if and only if the prediction loss of $x$ or any label preserving realistic variations ($x'$) is not greater than a *loss threshold*.

For selective-augmentation, Sensei-SA first determines whether $M$ is point-wise robust w.r.t. the seed. If the seed is robust, Sensei-SA does not augment it until the seed is incorrectly classified by $M$ in subsequent epochs or the prediction loss by $M$ is less than *loss threshold*. At any point, $M$ is unrobust w.r.t. the seed, Sensei-SA uses the optimal augmentation module to augment the seed.

## 4 EXPERIMENTAL SETUP

We evaluate Sensei with respect to three research questions:

- **RQ1** How effectively does Sensei improve the robustness of DNN models compared to state-of-the-art approaches?
- **RQ2** How effective the "selective augmentation" module in reducing the training time?
- **RQ3** How does the value of hyper-parameters affect the effectiveness and efficiency of Sensei?

### 4.1 Dataset and Models

Since computer vision is one of the most popular applications of deep learning, to evaluate our approach, we selected a wide range of image classification datasets described in Table 1. These datasets cover various applications such as traffic sign classification, object recognition, and age/gender prediction. Furthermore, all of these datasets have been widely used to evaluate training algorithms, adversarial attack and adversarial defense techniques. For each dataset, columns *#Train*, *#Test*, and *#CL* in Table 1 show the number of training, testing images, and the number of classes, respectively.

For each dataset, we collected multiple models (Column *#MD*) from open-source repositories. More specifically, we selected four models for *GTSRB* from [28, 38, 39], three models from [28, 36, 37] for *Fashion-MNIST (FM)*. The models for *CIFAR-10* include Wide-Resnet[44] and three Resnet[13] models with 20, 32, 50 layers, respectively, which are collected from [33]. For *SVHN*, we used a VGG model[28] and a model from [34]. As for *IMDB*, we consider two models: VGG16 and VGG19 [28] [35]. Except augmenting the training data, we do not change the original model architectures. All the detailed parameters can be found in repository of Sensei [1].

### 4.2 Generation of Realistic Variations

Sensei focuses on improving the robustness of DNN models by augmenting training data with natural environmental variations. Since, we focus on the applications with image, we choose two major kinds of image operations: i) geometric operations ii) color operations to simulate the camera movements and lighting conditions in real life. To make sure the translated images are visually similar to natural ones, we restrict the space of allowed perturbations following [8] where it is applicable. The operations and corresponding restrictions with respect to an image $x$ are as follows:

- *rotation(x,d)*: rotate $x$ by $d$ degree within a range [-30, 30].
- *translation(x, d)*: horizontally or vertically translate $x$ by $d$ pixels within a range of [-10%, 10%] of image size.
- *shear(x,d)*: horizontally shear $x$ with a shear factor $d$ within a range of [-0.1, 0.1].
- *zoom(x,d)*: zoom in/out $x$ with a zoom factor $d$ ranging [0.9,1.1]
- *brightness(x,d)*: uniformly add or subtract a value $d$ for each pixel of $x$ within a range of [-32, 32]
- *contrast(x,d)*: scale the RGB value of each pixel of $x$ by a factor $d$ within range of [0.8, 1.2]

These image operations preserve the content of original image. Since images in these datasets do not have any information about the pixels outside their boundary, the space beyond the boundary is assumed to be constant 0 (black) at every point.

### 4.3 Evaluation Metric

Since Sensei is focused on improving the robustness of DNN models, following Engstrom et al. [8], we compute robust accuracy of Sensei to answer each research question. More specifically, robust accuracy is the proportion of images in the testing dataset where the prediction of a DNN does not fluctuate with any small realistic perturbations. Formally, let us assume that there is an image $x$ in the testing dataset that belongs to a class $c$. $TS$ is a set of parametric transformations with a size of $m$. Applying a transformation $ts \in TS$ on $x$ gives us a transformed image $x'$. $X'$ is the set of all transformed images resulting from $TS$. So $|TS| = |X'|$. A DNN is robust around $x$ if and only if $M(x') = c$ for all $x' \in X'$. Finally, let us assume that *nInstances* is the number of images in the testing dataset, and among them *nRobustInstances* is the number of images where $M$ is robust. Then the robust accuracy of $M$ for the dataset is:

[1] https://sensei-2020.github.io

$$robust\ accuracy = \frac{nRobustInstances}{nInstances} \quad (6)$$

# 5 EXPERIMENTAL RESULTS

## 5.1 Implementation

We implement SENSEI on top of Keras version 2.2.4 (https://keras.io), which is a widely used platform that provides reliable APIs for training and testing DNNs. More specifically, we implement a new data generator that augments the training data during training. Our data generator takes as inputs the current model and original training set, augments the original data and then feeds the augmented data to the training process at each step. The augmented data is generated and selected using the approach described in Section 3.

## 5.2 Experimental Configurations

We conducted all the experiments on a machine equipped with two Titan V GPUs and Xeon Silver 4108 CPU 128G memory and 16.04 Ubuntu. All the experiment specific configurations are described in the respective answers. Since genetic algorithm in SENSEI involves random initialization and decision, we ran each experiment five times independently and reported the arithmetic average results.

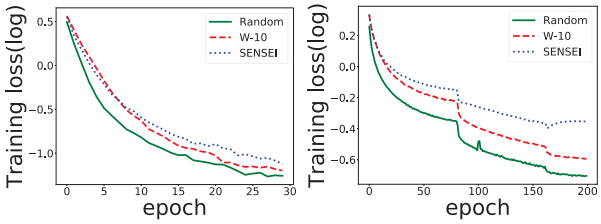
## 5.3 RQ1: Effectiveness of SENSEI

We perform a comprehensive set of experiments to evaluate the effectiveness of SENSEI compared to the state-of-the-art data augmentation approaches from various perspectives.

*5.3.1 Exp-1: Does SENSEI solve the saddle point problem effectively?* As we explained in Section 3.1, the effectiveness of a data augmentation technique lies in how effectively it solves the inner maximization of the saddle point problem in Equation 4. Therefore, in our first experiment, we check whether SENSEI is indeed effective in finding the most lossy variants effectively than the state-of-the-art techniques. To this end, we trained each model following three data augmentation strategies:

(1) **Random augmentation**. This is one of the most frequently used data augmentation strategies in practice since it is a built-in feature in the Keras framework. In this approach, given a set of perturbations, a random perturbation is performed for each image at each step (epoch). However, to make the comparison fair we customize the approach to give it the same combination of transformations as in SENSEI.
(2) **W-10**. The most recent data augmentation approach for natural variants, which is called Worst-of-10 [8]. W-10 generates ten perturbations randomly for each image at each step, and replaces the original image with the one on which the model performs worst [8] (e.g. highest loss).
(3) **SENSEI**. To make the comparison fair with W-10, the results of SENSEI are generated using a population size of 10.

**Results.** Figure 4 presents the logarithmic training loss for two models: GTSRB-1 and CIFAR-10-1. The results show that although SENSEI starts with very similar performance in the initial epochs, due to the systematic nature of SENSEI, soon it outperforms W-10 for every model and dataset. Therefore, the genetic algorithm based data selection in SENSEI is more effective to solve the inner maximization problem than *Random* and *W-10* based techniques.




**(a) GTSRB-1** **(b) CIFAR-10-1**

**Figure 4: Effectiveness in identifying most lossy variants in two models GTSRB-1 and CIFAR-10-1 (under *T3*)**

*5.3.2 Exp-2: Does SENSEI perform better than the state-of-the-art data augmentation techniques in any number of transformations?* It is harder to achieve robust accuracy as the number of transformation operators increases since there are just more options to fool the model. Therefore, we further investigate how the effectiveness of SENSEI vary as the number of transformations increases. We calculate robust accuracy under three (T3) and six image transformation operators (T6) separately. T3 experimentation includes the rotation, translation, and shear image operations as defined in Section 4.2.

**Results.** Table 2 presents core results of the paper, which shows the robust accuracy of all the models trained using the *Standard*, *Random*, *W-10* and SENSEI strategy using three and six transformation operators. From the results using T3, we see that even though the *Standard* training achieves over 91% average standard accuracy (shown in column *TestAcc*), the robust accuracy sharply drops to 5% on average (Column 4). *Random* augmentation and *W-10* based training significantly improve the robust accuracy for each dataset. However, SENSEI achieves the highest robust accuracy for all models of all data-sets (highlighted). SENSEI improved the robust accuracy from 8.2% to 18.7% w.r.t. random augmentation and from 1.7% to 6.1% w.r.t. state-of-the-art W-10. When we increased the number of transformation operators from three (T3) to six (T6), we see that the robust accuracy for all augmentation strategies decreased significantly. This is expected due to two facts: i) under *T6*, the generated variants are less similar with original images and ii) under *T6*, a larger number of perturbations are generated for each image, and an image is more likely to be considered as misclassified, since an image will be regarded as misclassified if one of its perturbation is misclassified. However, in this harder problem, the improvement by SENSEI compared to both random augmentation and W-10 is greater than that of T3. On average, SENSEI achieves 22.2% higher robust accuracy than *Random*, and 6.6% than *W-10*. This also demonstrates that SENSEI performs better in larger search space. Please note that we do not evaluate the models designed for IMDB with six transformation operators, because face image is very sensitive to the change of color palette.

*5.3.3 Exp-3: Does SENSEI perform better than the adversarial example-based retraining approaches?* In Section 2.4, we briefly described how data augmentation can be performed *during initial training vs. adversarial retraining*. The effectiveness of SENSEI

**Table 2: The robust accuracy for Random, W-10 and Sensei. Sensei uses *loss-based fitness* function.**

| Model | Size(MB) | TestAcc | Robust accuracy under 3 trans. op. (*T3*) | | | | Robust accuracy under 6 trans. op.(*T6*) | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Standard | Random | W-10 | Sensei | Standard | Random | W-10 | Sensei |
| GTSRB-1 | 16 | 98.00 | 3.20 | 77.60 | 85.80 | **90.80** | 2.10 | 49.90 | 64.00 | **67.90** |
| GTSRB-2 | 258 | 95.40 | 2.40 | 70.80 | 84.60 | **86.30** | 1.60 | 44.70 | 57.30 | **62.10** |
| GTSRB-3 | 11 | 97.92 | 0.70 | 72.10 | 83.30 | **88.70** | 0.60 | 42.40 | 56.70 | **64.40** |
| GTSRB-4 | 114 | 95.31 | 1.60 | 72.80 | 82.40 | **86.90** | 1.10 | 35.10 | 58.10 | **65.40** |
| FM-1 | 305 | 92.80 | 0.20 | 65.70 | 79.20 | **83.60** | 0.00 | 39.90 | 64.20 | **70.50** |
| FM-2 | 21 | 92.70 | 0.30 | 60.20 | 72.20 | **78.00** | 0.00 | 39.70 | 61.90 | **68.80** |
| FM-3 | 6 | 92.79 | 0.20 | 63.10 | 73.90 | **77.40** | 0.00 | 38.20 | 57.60 | **65.40** |
| CIFAR-10-1 | 7 | 88.32 | 1.30 | 52.20 | 61.80 | **67.20** | 0.10 | 33.10 | 47.10 | **55.20** |
| CIFAR-10-2 | 279 | 88.54 | 1.40 | 56.70 | 64.90 | **67.90** | 0.10 | 35.30 | 48.80 | **56.20** |
| CIFAR-10-3 | 5 | 88.01 | 1.80 | 73.30 | 76.50 | **81.50** | 0.40 | 57.40 | 70.60 | **72.50** |
| CIFAR-10-4 | 3 | 87.09 | 1.10 | 47.50 | 60.10 | **66.20** | 0.04 | 28.20 | 44.30 | **54.70** |
| IMDB-1 | 88 | 85.81 | 28.00 | 69.00 | 71.90 | **79.60** | - | - | - | - |
| IMDB-2 | 126 | 84.97 | 29.30 | 74.10 | 81.90 | **83.90** | - | - | - | - |
| SVHN-1 | 7 | 94.01 | 0.40 | 75.20 | 83.60 | **85.50** | 0.20 | 72.90 | 82.40 | **84.00** |
| SVHN-2 | 29 | 92.82 | 0.70 | 56.00 | 67.80 | **74.90** | 0.40 | 52.10 | 58.90 | **70.80** |
| AVG | - | 91.63 | 4.84 | 65.75 | 75.33 | **79.89** | 0.51 | 43.76 | 59.38 | **65.99** |

lies in that it effectively selects the optimal variation of the seed data, epoch by epoch. In contrast, in retraining based approach, once the adversarial examples are selected, they are fixed across epochs. In this experiment, we compare these two approaches. To this end, following the popular retraining approaches [15, 24], we replicated the adversarial training in a benign setting. The step includes: i) training the model using the original training data, ii) generating the adversarial examples by our transformations, i.e., the variants that fool the DNNs, iii) selecting the best adversarial examples, adding in the training data, and retraining the model for 5 additional epochs. To make the comparison fair, we generate the equal number of variants in Step-2 as of Sensei. For example, if Sensei generates 10 variants per data-point in one epoch and runs 100 epochs, we generate 1,000 adversarial examples for a given data-point, and choose the best variant of each data by the loss function. We still use the attack model and evaluation metric shown in Section 4.3.

**Table 3: Average robust accuracy by Sensei vs. Retraining**

| Approach | GTSRB | FM | CIFAR-10 | IMDB | SVHN |
|---|---|---|---|---|---|
| Retraining | 78.26 | 68.24 | 43.72 | 70.21 | 68.30 |
| Sensei | 88.18 | 79.67 | 70.70 | 81.75 | 80.20 |

**Results.** Table 3 presents the average robust accuracy of all models for Sensei and the retraining based approach. The results show that Sensei improves the robust accuracy from 10% to 27% compared to adversarial retraining, and the results are consistent across all datasets. Therefore, although adversarial retraining is very effective in a security-aware setting, Sensei is more effective in improving robust generalization in a benign setting.

*5.3.4 Exp-4: Can Sensei preserve the standard accuracy while improving robust generalization?* Improving the robust accuracy would only add value if Sensei can retain the standard accuracy. Therefore, we investigate how Sensei performs in terms of standard accuracy.

**Results.** Table 4 presents the average standard accuracy of the experimented models without data augmentation (2nd column) and trained by Sensei (4th column). Results show that the original (Standard) models are highly accurate. After we augmented each model by Sensei to increase their robust generalization, the standard accuracy increased even further for four out five datasets.

*5.3.5 Exp-5: How does Sensei perform compared to other state-of-the-art generalization techniques? mixup* [46] is a recent data augmentation approach that improves standard generalization and also guards against pixel-level perturbations in security-aware settings. *mixup* trains a neural network on convex combinations of pairs of examples and their labels to favor simple linear behavior in-between training examples. mixup's source-code to replicate the results for CIFAR-10 is available online [46]. We adapted mixup in our setting, verified with CIFAR-10 that our result is consistent with [46], and ran it for all the models and dataset in our study.

**Table 4: Average accuracy of mixup vs. Sensei**

| Models | Standard accuracy | | | Robust accuracy | | |
|---|---|---|---|---|---|---|
| | Stand. | mixup | Sensei | Stand. | mixup | Sensei |
| GTSRB | 96.65 | 97.24 | 97.43 | 1.98 | 1.72 | 88.18 |
| FM | 92.77 | 92.95 | 89.53 | 0.23 | 1.69 | 79.67 |
| CIFAR-10 | 87.98 | 92.45 | 90.93 | 1.40 | 1.69 | 70.70 |
| IMDB | 85.40 | 89.51 | 88.30 | 28.65 | 30.23 | 81.75 |
| SVHN | 93.40 | 95.94 | 94.60 | 0.55 | 0.25 | 80.20 |

**Results.** Table 4 compares the average robust accuracy achieved by mixup and Sensei both for standard generalization and robust generalization across all models. The results show that mixup and Sensei both overall improved standard generalization. In fact, mixup is a little better than Sensei for standard generalization. However, mixup performed poorly in improving robust generalization for real-world naturally occurring variants. It performed

marginally better than no-augmentation. In contrast, Sensei clearly outperformed mixup for every dataset for robust generalization.

**Comparison with Yang et al. [42].** Very recently, Yang et al. have proposed an approach to increase DNN robustness to spatial transformations, by modifying the loss function of the DNN. Specifically, they retain the random W-10 augmentation strategy [8] but add an invariant-inducing regularization term to the standard empirical loss, to boost robustness. However, their technique is implemented in a different framework (TensorFlow, versus Keras for Sensei) and their experiments are performed on significantly different DNN models and with different transformations (only 2 versus up to 6 used by Sensei) and evaluation metrics. This makes a fair, head-to-head quantitative comparison infeasible at present. However, for an initial assessment of relative performance, we ran Sensei on CIFAR-10, with a model *substantially similar* to one used in [42], and using the same two transformations. The results show that Sensei reduces the error (the evaluation metric used in [42]) by 23.5% over W-10 while their technique reduced it by 21.9%, as reported in [42]. Thus, Sensei can match or surpass their technique in this case.

Conceptually, Sensei and [42] improve DNN robustness through fundamentally different mechanisms. Sensei uses a data-augmentation approach – a black-box technique, while Yang et al. use a re-designed loss function – a white-box approach. These techniques are thus complementary and combining them for even greater robustness improvement could be interesting future work.

> Sensei solves the inner maximization problem more effectively than the state of the art. It is able to improve the robust accuracy of the DNN, compared to the state of the art, on each of the 15 models, by upto 11.9%, while retaining standard test accuracy. Furthermore, for benign variations, Sensei outperforms data augmentation approaches both that are based on adversarial retraining or that target standard generalization.

## 5.4 RQ2: Effect of Selective Data Augmentation

We analyze the performance of Sensei-SA in terms of (i) training time and (ii) robust accuracy compared to Sensei and W-10. Note that the standard training requires only one forward propagation for calculating the loss and one backward propagation for calculating gradients to update the weights per data-point at each epoch. However, the training time with data augmentation includes the additional time for optimal selection. Specifically, Sensei requires $N$ additional forward propagation to calculate the fitness of $N$ newly generated population (same for *W-10* to find the worst case). Following the standard protocol [16], we do not count the image transformation time because these tasks are completed on CPU, which is executed in parallel with the training task processed on GPU. For this experiment, we set *loss threshold* in Sensei-SA to $1e{-}3$ (described in Section 3.4).

Table 5 summarizes the robust accuracy and training time for all the dataset. Column *Improvement* represents the improvement achieved by Sensei-SA compared to *W-10* and the last column presents Cohen's D values [5] to show the effect size of improvements over the five runs. The evaluation results indicate that Sensei-SA significantly improve robust accuracy over *W-10*. From the

**Table 5: Sensei-SA vs. W-10 and Sensei**

| Models | Robust accuracy / Training time(min) | | | | |
|---|---|---|---|---|---|
| | W-10 | Sensei | Sensei-SA | Improve | D-value |
| GTSRB | 84.03 / 18.4 | 88.18 / 18.8 | 86.13 / 11.9 | 2.10 / 36% | 6.3 |
| FM | 75.10 / 28.2 | 79.67 / 29.3 | 78.63 / 20.9 | 3.53 / 21% | 25.6 |
| CIFAR-10 | 65.83 / 192 | 70.70 / 158 | 69.40 / 120 | 3.57 / 34% | 35.0 |
| IMDB | 76.90 / 946 | 81.75 / 962 | 81.09 / 952 | 4.19 / (1%) | 22.3 |
| SVHN | 75.70 / 58.8 | 80.20 / 58.9 | 79.65 / 51.5 | 3.95 / 13% | 42.5 |

results, we can also see that with the help of selective augmentation, Sensei-SA reduced the training time by 25% on average. The training time of models for *IMDB* is not significantly reduced, because predicting the age of a person is very hard even for human, and only very few data points are robust enough to be excluded from augmentation. It should be noted that although Sensei-SA selectively augmented the data points, on average, the robust accuracy achieved by Sensei-SA is 3% higher than that of *W-10*. However, compared to Sensei, Sensei-SA reduced the robust accuracy by 0.5-1.5% for most of the models.

> On average, Sensei-SA reduces the training time by 25%, while it improves robust accuracy by 3% compared to W-10.

## 5.5 RQ3: Sensitivity of Hyper-Parameters

In this section, we evaluate how the choice of different hyper-parameters influence the performance of Sensei.

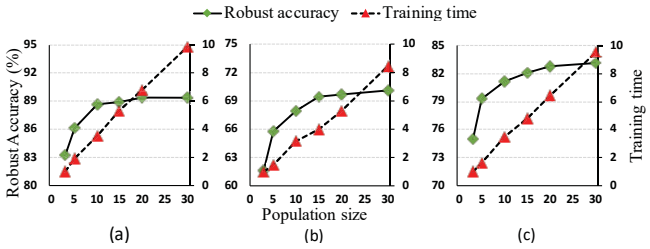



**Figure 5: Robust accuracy and normalized training time for one model in (a) GTSRB (b) CIFAR-10 and (c) FM by Sensei**

*5.5.1 Population size.* Since population size is an important parameter in any genetic search [26], we evaluate Sensei using different population size. Figure 5 presents the robust accuracy and training time of three models trained by Sensei with population size ranging (3, 30). The training time in the graph is normalized compared to the training time of population size of 3. The results show that the robust accuracy in each model increases with the increase of population size. However, a high population size also impacts the training time negatively. Results show that Sensei works efficiently when the population size is between 10 and 15. Further increase of the population size does not improve the robust accuracy a lot.

> Sensei works efficiently between the population size 10 and 15 for both robust accuracy and training time.

**Table 6: The robust accuracy of SENSEI with *loss-based* and *coverage-based fitness function***

| Fitness | GTSRB | FM | CIFAR-10 | IMDB | SVHN |
|---|---|---|---|---|---|
| Loss-based | 88.18 | 79.67 | 70.70 | 81.75 | 80.20 |
| Neuron Coverage | 85.17 | 79.32 | 69.90 | 81.40 | 80.43 |

*5.5.2 Fitness function.* The efficiency of a genetic search depends a lot on the fitness function. Although we evaluated SENSEI rigorously using *loss-based* fitness function, as we discussed in Section 3.3.3, other metrics such as *neuron coverage-based* [24] or *surprise adequacy* [15] can be also used as a fitness function. Table 6 presents the robust accuracy of SENSEI for neuron coverage based fitness function. The results show that both loss-based and coverage-based fitness functions achieve very similar robust accuracy. Since surprise adequacy is correlated to neuron coverage [15], we expect a similar performance also using surprise adequacy. However, loss based fitness function may be a better choice since neuron coverage based fitness function increases the training time by 50% than that of loss-based fitness function. The reason is that computation of neuron coverage is more expensive than training loss.

> The state-of-the-art DNN quality metrics such as neuron coverage show similar performances in terms of robust accuracy and thus, can be used as a fitness function in SENSEI for robust generalization. However, *loss-based* fitness function may be a better choice due to shorter training time.

**Table 7: Effect of selection metric in SENSEI-SA in terms of average robust accuracy (loss threshold: 1e-3)**

| Metrics | GTSRB | FM | CIFAR-10 | IMDB | SVHN |
|---|---|---|---|---|---|
| Loss-based | 86.13 | 78.63 | 69.40 | 81.09 | 79.65 |
| Classification-based | 79.08 | 64.33 | 49.35 | 63.00 | 63.42 |

*5.5.3 Selection metrics.* The performance of SENSEI-SA in terms of both robust accuracy and training time depends on the effectiveness of the *point-wise robustness* metric (defined in Section 3.4). The evaluation results in Table 7 show that the loss-based selection outperforms the classification-based selection for all the models. The reason is that the loss-based selection is more conservative than the classification-based selection. Still, loss-based selection is good enough to skip sufficient number of data-points to achieve 25% training time reduction, on average.

> *Loss-based robustness* is better than *classification-based robustness* in selective augmentation.

*5.5.4 Loss threshold.* In *loss-based* selection, the loss threshold is one of the important factors that may affect the effectiveness of SENSEI-SA. Figure 6 shows robust accuracy and normalized training time of SENSEI-SA with loss threshold in range (0, 1e−1). The training time is normalized to *Standard* training time. From the results,

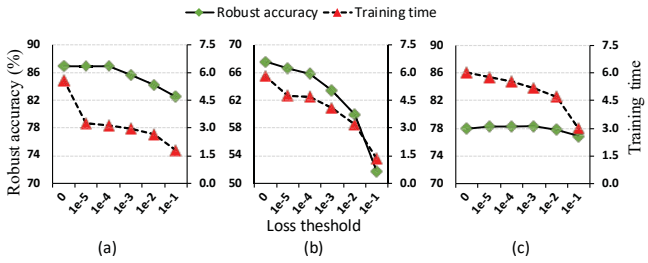



**Figure 6: The robust accuracy and normalized training time of one model from (a) GTSRB (b) CIFAR-10 (c) FM trained by SENSEI-SE with various loss thresholds in range (0, 1e−1).**

as expected we observe that both the robust accuracy and training time decrease with the increase of loss threshold. However, some datasets are more sensitive to the loss threshold than the others in terms of robust accuracy. For instance, the robust accuracy for the CIFAR-10 model is very sensitive to the loss threshold. However, robust accuracy of GTSRB and FM models did not decrease a lot when we changed the loss threshold from 1e-5 to 1e-3.

> The effect of loss threshold in selective augmentation is dataset specific. However, a value of 1e-3 showed a balanced performance across datasets in terms of robust accuracy.

## 5.6 Threats to Validity

*Internal validity:* We have tried to be consistent with established practice in the choice and application of image transformations, and training schedule for DNNs. For parameters specific to our technique, including population size and fitness function for GA, selection function and loss threshold for selective augmentation we have performed a sensitivity analysis to justify the claims.

*External validity:* Although we used 5 popular image datasets, and several DNN models per dataset in our evaluation, our results may not generalize to other datasets, or models, or for other applications. Further, our experiments employ six commonly used image transformations, to model natural variations. However, our results may not generalize to other sources of variations.

# 6 RELATED WORK

A DNN can be viewed as a special kind of program. Software engineering techniques for automated test generation, as well for test-driven program synthesis and repair, have either directly inspired or have natural analogs in the area of DNN testing and training. The contributions of these techniques relative to ours can be compared in terms of the following four facets.

**Test adequacy metrics.** Inspired by structural code coverage criteria, Pei et al. [24] proposed Neuron Coverage to measure the quality of a DNN's test suite. DeepGauge [19] built upon this work and introduced a number of finer-grained adequacy criteria, including $k$-Multisection Neuron Coverage and Neuron Boundary Coverage. Kim et al. [15] also proposed a metric called surprise adequacy to select adversarial test inputs. MODE [20] performs state differential analysis to identify the buggy features of the model and

then performs training input selection on this basis. Our contribution is orthogonal to these test selection criteria. We demonstrate how to instantiate our technique with either standard model loss or neuron coverage. In principle, Sensei could be adapted to use other criteria as well.

**Test generation technique.** The earliest techniques proposed for DNN testing were in a security setting, as *adversarial testing*, initially for computer vision applications. Given an image, the aim is to generate a variant, with a few pixels selectively modified – an *adversarial instance* – on which the DNN mis-predicts. Such techniques include [29], FGSM [12], JSMA [23] and so on. At a high level, these approaches model the generation of adversarial examples as an optimization problem and solve the optimization problem using first-order methods (gradient ascent). Generative machine learning, such as GAN [11], can also be used to generate adversarial inputs. In contrast to the above techniques, our focus is the space of benign natural variations, such as rotation and translation. Engstrom et al. showed that such transformations yield a non-convex landscape not conducive to first-order methods [8]. Thus, our test generation technique uses fuzzing, based on genetic search. Recently, Odena and Goodfellow proposed TensorFuzz [22] that combines coverage-guided fuzzing with property-based testing to expose implementation errors in DNNs. However, their coverage metric, property oracles, and fuzzing strategies are all designed around this specific objective and not suitable for our objective of data augmentation driven robustness training. DeepXplore [24] generates test inputs that lead to exhibit different behaviors by different models for the same task. Our approach does not require multiple models. DeepTest [30] and DeepRoad [47] use metamorphic testing to generate tests exposing DNN bugs in the context of an autonomous driving application. While Sensei's search space is also defined using metamorphic relation, the mode of exploring the search space (genetic search) and incorporating them (data augmentation) is fundamentally different from these techniques.

**Test incorporation strategy.** The vast majority of DNN test generation techniques [14, 15, 17, 24, 30, 31] first use a trained DNN to generate the tests (or adversarial instances) and then use them to *re-train* the DNN, to improve its accuracy or robustness. By contrast, Sensei falls in the category of data augmentation approaches where new test data is generated and used during the initial DNN training. As shown in our evaluation our data augmentation yields better robustness compared to the former generate and re-train approach.

Data augmentation can be performed with different objectives. AutoAugment [6] uses reinforcement learning to find the best augmentation policies in a search space such that the neural network achieves the highest (standard) accuracy. Mixup [46] is a recently proposed state-of-the-art data augmentation technique that trains a neural network on convex combinations of pairs of examples (such as images) and their labels. Our evaluation (Section 5) confirms that mixup improves both standard accuracy and robust accuracy in a security setting but performs poorly in terms of robust accuracy in a benign setting. By contrast, Sensei, with a completely different search space and search strategy excels in this area.

Our work is inspired by the theoretical work of Madry et al. [21] who formulated the general data augmentation problem as a saddle point optimization problem. Our work instantiates a practical solution for that problem in the context robustness training for benign natural variations by using a genetic search naturally overlaid on the iterative training procedure for a DNN. The work closest to ours is the one by Engstrom et al. [8] who were the first to show that benign natural image perturbations, notably rotation and translation, can easily fool a DNN. They proposed a simple data augmentation approach randomly sampling $N$ perturbations and replacing the training example with the one with the worst loss. Our approach improves on both the robust accuracy as well as the training time of Engstrom's approach, by using a systematic genetic search to iteratively find the worst variant to augment and using a local robustness calculation to save the augmentation and training time.

**Robust models.** Recently, Yang et al. [42] proposed an orthogonal approach to improve the robustness deep neural network models by modifying the DNN loss function and adding an invariant-inducing regularization term to the standard empirical loss. Conceptually, this regularization based white-box approach is complementary to our black-box approach of data augmentation. Combining the two approaches, for even greater robustness improvement, could be interesting future work.

## 7 CONCLUSION

Recent research has exposed the poor robustness of DNNs to small perturbations to their input. A similar lack of generalizability manifests, as the over-fitting problem, in the case of test-based program synthesis and repair techniques where test generation techniques have recently been successfully employed to augment existing specifications of intended program behavior. Inspired by these approaches, in this paper, we proposed Sensei, a technique and tool that adapts software testing methods for data augmentation of DNNs, to enhance their robustness. Our technique uses genetic search to generate the most suitable variant of an input data to use for training the DNN, while simultaneously identifying opportunities to accelerate training by skipping augmentation, with minimal loss of robustness. Our evaluation of Sensei on 15 DNN models spanning 5 popular image datasets shows that, compared to the state of the art, Sensei is able to improve the robust accuracy of the DNNs by upto 11.9% and on average 5.5%, while also reducing the DNN's training time by 25%.

Since significant amount of decision making in public-facing software systems are being accomplished via deep neural networks, reasoning about neural networks has gained prominence. Instead of developing verification or certification approaches, this paper has espoused the approach of data augmentation via test generation to improve or repair hyper-properties of deep neural networks. In a broader sense, this work also serves as an example of harnessing the rich body of work on testing, maintenance, and evolution for traditional software, for developing AI-based software systems.

## ACKNOWLEDGMENTS

This work was partially supported by the National Satellite of Excellence in Trustworthy Software Systems, funded by NRF Singapore under National Cybersecurity R&D (NCR) programme.

# REFERENCES


[1] Rajeev Alur, Rishabh Singh, Dana Fisman, and Armando Solar-Lezama. 2018. Search-based Program Synthesis. *Commun. ACM* 61 (2018).

[2] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. 2009. *Robust optimization*. Vol. 28. Princeton University Press.

[3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *ArXiv preprint arXiv:1604.07316* (2016).

[4] Nicholas Carlini and David Wagner. 2017. Magnet and "efficient defenses against adversarial attacks" are not robust to adversarial examples. *ArXiv preprint arXiv:1711.08478* (2017).

[5] Jacob Cohen. 2013. *Statistical power analysis for the behavioral sciences*. Routledge.

[6] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. 2019. AutoAugment: Learning Augmentation Strategies From Data. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[7] Pedro M Domingos. 2012. A few useful things to know about machine learning. *Communication of the ACM* 55, 10 (2012), 78–87.

[8] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. 2019. Exploring the Landscape of Spatial Robustness. In *International Conference on Machine Learning (ICML)*. 1802–1811.

[9] Yu Feng, Ruben Martins, Osbert Bastani, and Isil Dillig. 2018. Program synthesis using conflict-driven learning. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. ACM, 420–435.

[10] Xiang Gao, Sergey Mechtaev, and Abhik Roychoudhury. 2019. Crash-avoiding Program Repair. In *ACM SIGSOFT International Symposium on Testing and Analysis (ISSTA)*.

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*. 2672–2680.

[12] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[14] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. 2011. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*. ACM, 43–58.

[15] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding deep learning system testing using surprise adequacy. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*. IEEE Press, 1039–1049.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*. 1097–1105.

[17] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2017. Adversarial machine learning at scale. In *International Conference on Learning Representations (ICLR)*.

[18] Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. 2019. Automated Program Repair. *Commun. ACM* 62, 12 (2019).

[19] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. 2018. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)*. ACM, 120–131.

[20] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. 2018. MODE: automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 175–186.

[21] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*.

[22] Augustus Odena and Ian Goodfellow. 2018. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. In *International Conference on Machine Learning (ICML)*.

[23] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 372–387.

[24] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*. ACM, 1–18.

[25] Zichao Qi, Fan Long, Sara Achour, and Martin Rinard. 2015. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In *International Symposium on Software Testing and Analysis (ISSTA)*.

[26] O. Roeva, S. Fidanova, and M. Paprzycki. 2013. Influence of the population size on the genetic algorithm performance in case of cultivation process modelling. In *2013 Federated Conference on Computer Science and Information Systems*. 371–376.

[27] Patrice Y Simard, David Steinkraus, John C Platt, et al. 2003. Best practices for convolutional neural networks applied to visual document analysis.. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition(ICDAR)*, Vol. 3.

[28] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*.

[29] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*.

[30] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering (ICSE)*. ACM, 303–314.

[31] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2018. Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations (ICLR)*.

[32] Website. 2019. American Fuzzy Lop (AFL). http://lcamtuf.coredump.cx/afl Accessed: 2019-04-08.

[33] Website. 2019. Cifar-10. https://github.com/BIGBALLON/cifar-10-cnn. Accessed: 2019-03-10.

[34] Website. 2019. Cifar-10. https://github.com/yh1008/deepLearning. Accessed: 2019-03-10.

[35] Website. 2019. Cifar-10. https://github.com/abars/YoloKerasFaceDetection. Accessed: 2019-03-10.

[36] Website. 2019. Fashion-MNIST. https://github.com/umbertogriffo/Fashion-mnist-cnn-keras. Accessed: 2019-03-10.

[37] Website. 2019. Fashion-MNIST. https://github.com/markjay4k/Fashion-MNIST-with-Keras. Accessed: 2019-03-10.

[38] Website. 2019. GTSRB. https://github.com/chsasank/Traffic-Sign-Classification.keras. Accessed: 2018-10-30.

[39] Website. 2019. GTSRB. https://github.com/xitizzz/Traffic-Sign-Recognition-using-Deep-Neural-Network. Accessed: 2018-10-30.

[40] Qi Xin and Steven P Reiss. 2017. Identifying test-suite-overfitted patches through test case generation. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 226–236.

[41] Yingfei Xiong, Xinyuan Liu, Muhan Zeng, Lu Zhang, and Gang Huang. 2018. Identifying patch correctness in test-based program repair. In *Proceedings of the 40th International Conference on Software Engineering (ICSE)*. ACM, 789–799.

[42] Fanny Yang, Zuowen Wang, and Christina Heinze-Deml. 2019. Invariance-inducing regularization using worst-case transformations suffices to boost accuracy and spatial robustness. In *Advances in Neural Information Processing Systems (NIPS)*. 14757–14768.

[43] Zhongxing Yu, Matias Martinez, Benjamin Danglot, Thomas Durieux, and Martin Monperrus. 2018. Alleviating patch overfitting with automatic test generation: a study of feasibility and effectiveness for the Nopol repair system. *Empirical Software Engineering* (2018), 1–35.

[44] Sergey Zagoruyko and Nikos Komodakis. 2016. Wide Residual Networks. In *British Machine Vision Conference (BMVC)*.

[45] Valentina Zantedeschi, Maria-Irina Nicolae, and Ambrish Rawat. 2017. Efficient defenses against adversarial attacks. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 39–49.

[46] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. 2017. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations (ICLR)*.

[47] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)*. 132–142.

[48] Wenyi Zhao, Rama Chellappa, P Jonathon Phillips, and Azriel Rosenfeld. 2003. Face recognition: A literature survey. *ACM computing surveys (CSUR)* 35, 4 (2003), 399–458.