2019 IEEE International Conference on Autonomic Computing (ICAC)

Enhancing Learning-Enabled Software Systems to Address Environmental Uncertainty

Michael Austin Langford and Betty H.C. Cheng
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan, USA
{langfo37, chengb}@cse.msu.edu

***Abstract*—An overarching problem with Learning-Enabled Systems (LES) is determining whether training data is sufficient to ensure the LES is resilient to environmental uncertainty and how to obtain better training data to improve the system's performance when it is not. Automated methods can ease the burden for developers by augmenting real-world data with synthetically generated data. We propose an evolution-based method to assist developers with the assessment of learning-enabled systems in environments not covered by available datasets. We have developed Enki, a tool that can generate various conditions of the environment in order to discover properties that lead to diverse and unique system behaviors. These environmental properties are then used to construct synthetic data for two purposes: (1) to assess a system's performance in an uncertain environment and (2) to improve system resilience in the presence of uncertainty. We show that our technique outperforms a random generation method when assessing the effect of multiple adverse environmental conditions on a Deep Neural Network (DNN) trained for the commonly-used CIFAR-10 benchmark.**

***Index Terms*—machine learning, artificial neural networks, evolutionary computation, novelty search, search-based software engineering, software assurance, uncertainty**

I. Introduction

High assurance cyber-physical systems, such as autonomous vehicles, must overcome numerous challenges posed by environmental uncertainty when addressing assurance requirements [1]. With the inclusion of machine learning components, such as Deep Neural Networks (DNNs), these systems become learning-enabled. Verifying that the behavior of a Learning-Enabled System (LES) [2] matches expectations requires a selection of "good" test data that can be challenging to obtain. Tesla and Uber have both reported automobile accidents involving autonomous systems [3] [4] [5]. These systems must be able to handle a broad spectrum of scenarios when deployed, scenarios that may lead to situations beyond the training environment, where trustworthiness must be established at the design stage. This paper introduces an automated technique to discover scenarios of environmental uncertainty (e.g., environmental effects such as rain or lighting) that produce a wide range of behavior from an LES. These selected scenarios can then be leveraged to help assess and improve the system's performance in the face of adverse or uncertain environmental conditions.

Unlike traditional software, machine learning components typically lack an explicit internal logic and rely on intermediate representations of control data that are often obfuscated or difficult to interpret. The common practice to verify the effectiveness of a machine learning system is to evaluate how well it can make predictions on *test data*. However, selecting "good" data for this purpose is challenging [6] [7] [8]. Supervised learning systems learn from induction, taking specific examples from a set of *training data* to model more abstract target functions. Machine learning systems are said to perform well when they generalize, which is typically estimated by the system's performance on test data. Because each example of training and test data must be manually labeled, the cost of data production is expensive in terms of both time and labor. Manual selection of test data by software developers can also be susceptible to cognitive biases [9]. For large, high-dimensional problem spaces that are difficult to visualize, there is potential for relevant regions of the problem space to remain unrepresented in either the training or test data. For example, an autonomous vehicle that has only been trained and evaluated on clear, sunny days may not perform well in the presence of rain. By simulating uncertain environmental effects, computer-assisted techniques may be used to augment existing datasets to expand coverage of the problem space.

This paper describes an evolution-based technique to construct new, synthetic training and test data for an LES by generating various environmental effects and selecting those that cause the most unique and extreme system behavior. For example, an environmental effect may describe where and how a raindrop appears on an image from an autonomous vehicle's onboard camera, and the described technique can be used to discover a number of raindrop variations that impact the performance of the autonomous vehicle in mutually unique (i.e., *diverse*) ways. This technique uses an evolution-based search method that is able to generate simulated environmental effects on sensor data via parameterized *transformation functions*. Diverse environmental effects are discovered by iteratively refining the parameters of their associated transformation functions (e.g., raindrop radius, blur factor, etc.) such that each individual effect results in increasingly deviant system behavior, in relation to the other effects under consideration. The resulting transformation functions can then be used to construct synthetic data when applied to existing, unaltered data. This technique enables a developer to automatically assess a machine learning system under a range of environmental conditions not covered by existing data, and furthermore, it provides a means to train machine learning systems to be more robust and resilient to a wider range of environmental conditions.

We have developed Enki,[1] a blackbox automation tool that supports the proposed technique. Enki can be used for discovering diverse operating conditions for any general System Under Test (SUT), based on the SUT's response. This paper addresses the following research questions:

RQ1.) Can we use Enki to identify gaps in training that cause an LES to perform poorly?

RQ2.) Can we use Enki to improve the resiliency of an LES to the effects of uncertain environmental conditions?

To answer these questions, this paper focuses on the problem of image classification with DNNs in unfavorable environmental conditions, a key capability needed for several autonomous driving features (e.g., obstacle avoidance, lane management, and adaptive cruise control). Given its widespread use, we have implemented a DNN for the commonly-used CIFAR-10 benchmark [10] and conducted a number of experiments with different environmental effects to show how Enki can be used to construct useful synthetic test and training data for this task. Additionally, we compare our results to a random generation method and other approaches described by related work.

Our results show that by using Enki, we can generate environmental effects, such as decreased lighting, haze, and the presence of rain to construct a set of effects that negatively impact the performance of a CIFAR-10 DNN. Furthermore, we demonstrate that synthetic training data generated by Enki can then be used to improve the accuracy of the DNN in the presence of the same adverse environmental effects. The remainder of this paper is organized as follows. Section II overviews background information on DNNs and related work. Section III describes the framework and methodology of the proposed technique. Section IV includes results from an empirical evaluation of the technique. Finally, Section V provides a conclusion for this study.

II. Background and Related Work

This section provides a brief review of DNNs and challenges involved with their validation. Brief descriptions of automated testing techniques, novelty search, and related work in testing DNNs are also provided.

A. Deep Neural Networks

DNNs are applications of machine learning that are popular due to their potential to emulate any complex system with sufficient training. In the case of autonomous vehicles, they may be used to process data from onboard sensors [11] to assist with tasks such as lane-keeping [12] and collision-avoidance [13]. Represented with multi-layered architectures, as shown in Fig. 1, DNNs are composed of multiple intermediate *hidden layers* that connect a set of input *features*, x_i, to target output *labels*, y_i. Input features can include any observable property, such as pixels for images from camera or radar sensors. Output labels can describe any piece of information that may be inferred from the input, such as a safe steering angle or brake pressure for a vehicle. Each hidden layer, comprising a number of units called *neurons*, represents a single linear transformation of the layer's input followed by a non-linear *activation* function that acts as a mechanism to filter or amplify information derived from the layer's input. DNNs are trained to approximate target functions by adjusting weight parameters, $w_{i,j}$, corresponding to each neuron. Typically, weights are adjusted to minimize an *objective loss* function that measures the amount of error between a DNN's output and the ground truth. Once the objective loss is minimized, the training phase is terminated, and the DNN is verified by evaluating it against separate test data.

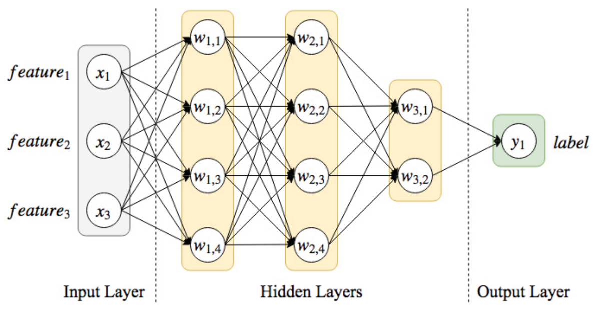

Fig. 1: High-level illustration of a simple DNN. Hidden layers allow the network to model a target function that maps input features to output labels. Neurons are represented by circles and links represent the flow of data from input to output.

By linking layers together, DNNs are capable of learning increasingly abstract relationships between features in the source input [14]. However, when weights are adjusted for the network as a whole with an objective defined only in terms of the network's final output, as with typical training methods, developers have no direct control over how any specific hidden layer infers information in isolation. As such, intermediate representations of information inferred by the DNN can be difficult to interpret. Even when the final output of a DNN may appear to be correct for all evaluated test data, the exact reasoning for how a DNN determines its output is often a mystery [15], though there is active research in methods to quantify the interpretability of hidden layers [16], visualize intermediate representations [17], and verify software implementations of an LES [18] [19] [20]. In contrast, this paper focuses on identifying gaps in training data that make the DNN less resilient to uncertainty and enhancing the DNN with synthetic training data.

B. Challenges for Deep Neural Networks

AlexNet [21], an image recognition application, popularized the use of DNNs for computer vision tasks by showing significant improvements over existing techniques. With these state-of-the-art DNNs, the number of hidden layers in a network's topology has been associated with an ability to

[1]Enki is a Sumerian god associated with knowledge, mischief, and creation.

learn higher levels of abstraction from features in the source input. Research even suggests that DNNs can compete with the human visual cortex [22]. However, this intuition has been challenged by the discovery of "adversarial examples" that exploit a DNN into making incorrect predictions in unexpected ways [23] [24] [25].

Adversarial examples have been shown to force a DNN to make false negative predictions by only adding small amounts of noise (imperceptible to a human) to images [26] and false positive predictions with high confidence for images constructed entirely from noise [27]. Furthermore, studies have shown that when both training and test data contain the same surface statistical regularities, it is possible for a DNN to learn and make predictions based on superficial details of an image's composition rather than the high-level semantics of the image's contents [28]. In such cases, a DNN can show a high degree of accuracy on test data while not successfully generalizing to data outside of the training or test datasets.

C. Search-Based Testing

Test case generation for software verification and validation has been a long-standing challenge, including considerable research in automated Search-Based Testing (SBT) techniques [29]. These search-based methods explore a space of candidate test cases to find a subset that optimizes a given objective (e.g., software coverage, program faults, etc.). [30]. The simplest approach is *random search*, where candidate test cases are randomly generated until a desirable set is found. More advanced techniques introduce heuristics to help guide the search towards relevant test cases more reliably than random generation [29]. Tools such as EvoSuite [31] and SAPIENZ [32] use Evolutionary Algorithms (EAs). These tools require software engineers to specify a test case representation that can be encoded by a *genotype* for the EA, where each encoding of a test case is referred to as a *genome*. The EA includes an iterative process of test case generation and selection to evolve better quality test cases. Genomes are manipulated through operations such as *crossover* and *mutation*, and selected for preservation between generations by a *fitness* heuristic. Fitness is typically a metric based on observable properties of the SUT, such as code coverage, which may be encoded by a *phenotype*. Thus, EAs explore the space of test cases by their genomes and compare the quality of test cases by their associated *phenomes*. Through this process, SBTs can use evolution to discover test cases pertinent to the SUT, relative to the desired fitness metrics.

D. Novelty Search

Novelty search has been described in detail by Lehman and Stanley [33] [34] as a method to search large candidate spaces for interesting results in the absence of an explicit objective. Greedy algorithms that focus on maximizing or minimizing an objective function have been found to be efficient and effective for simple problem spaces, but they are prone to discovering suboptimal solutions when the problem space contains many local optima or the global optima covers a very small region with no smooth gradient. In contrast, novelty search ignores any specific objective other than diversifying each candidate solution. Diversity is determined by comparing each candidate with its neighbors and favoring candidates with greater distance from their nearest neighbors. Thus, through an iterative process, novelty search can discover a collection of candidate solutions that cover a wide region of the problem space while preventing all candidates from being attracted to the same local optima. When implemented as an EA, a population of individual candidates can be evolved based on how diverse they are in relation to an archive of the most diverse candidates discovered.

Enki is inspired by Loki [35], an approach to integrate novelty search into a Self-Adaptive System (SAS) in order to generate environmental conditions that lead to diverse system behavior as specified by high-level goal models [36]. In contrast to Loki, Enki is a standalone system and is not limited to traditional code-based systems. Enki also does not require a goal model to be defined for the SUT. Enki provides a user flexibility and control over how novelty is computed to find operating conditions that diversify any specified set of performance metrics, beyond the scope of goal satisfaction for requirements engineering. More research has been conducted on the topic of using novelty search for automated testing, such as the work of Boussa et al. [37] [38] to generate test data for object-oriented systems. However, such approaches use novelty search to diversify test cases by their testing parameters rather than by the behavior they produce in the SUT. By using the SUT's behavior as the basis for diversity, Enki is able to discover test cases that may have similar operating conditions but unexpectedly lead to very different behavior.

E. Testing Deep Neural Networks

Two existing techniques have been proposed to tackle the problem of automatic test generation for DNNs: DeepXplore [39] and DeepTest [40]. DeepXplore treats data generation as a joint optimization problem with two objectives: maximizing *differential behavior* and maximizing *neuron coverage* for synthetically produced inputs. Differential behavior is determined by evaluating multiple DNNs trained on the same task with identical inputs and comparing the objective loss between DNNs. Neuron coverage is defined as the ratio of neurons activated while processing given inputs to the total number of neurons. DeepTest also generates data based on neuron coverage but requires only a single DNN in the process. DeepTest performs random transformations on given inputs to simulate environmental effects and greedily favors transformations that increase the neuron coverage for the target DNN. By maximizing neuron coverage, both methods aim to generate synthetic data that exercise the largest portion of the DNN.

Both DeepXplore and DeepTest demonstrate an ability to uncover weaknesses in DNNs. However, neither technique encourages diversity in terms of how a DNN performs under each test case. Enki differs from these techniques, since it explicitly searches for operating conditions that lead to unique system responses. Instead of only discovering test cases that highlight weaknesses in a DNN, Enki can be used to discover a broad range of test cases that may uncover strengths, weaknesses, or otherwise unknown (latent) behavior. Furthermore, Enki's emphasis on diversity not only identifies weaknesses but also identifies the most diverse weaknesses, thus enabling better coverage of vulnerabilities. Another key difference between these two techniques and this paper's application of Enki is in the form of each technique's output. Both DeepXplore and DeepTest generate transformations that are linked to a specific source input, and therefore, no general trends can be inferred from the synthetic data produced. In contrast, Enki generates transformations independent of any specific input, and therefore, more generalized patterns can be inferred from the results (see Section IV-B and Fig. 10) that inform the developer which range of conditions should be targeted for alternate strategies (e.g., include different sensors and/or add additional sensors).

III. Application Framework

The section provides an overview of Enki and how it can be used to assess and enhance a DNN in the presence of environmental uncertainty. Enki performs novelty search to automatically discover operational conditions that result in the most diverse system behavior of an SUT. The user must provide a specification for the operational conditions of interest, including the variables of uncertainty with corresponding ranges of values for Enki to explore. One example case of environmental uncertainty is how rain may affect an image-recognition DNN when raindrops partially occlude its view. This specific case study is considered to explain each component of the application process. However, this technique is not limited to only simulating raindrops. Fig. 2 contains a data flow diagram to visualize our approach, where bubbles represent processes, arrows represent data flow, parallel lines represent data stores, and boxes represent external entities. Steps 1 through 4 cover the first objective of assessing a DNN's performance in the presence of uncertain environmental conditions of interest (e.g., different representations of raindrops that vary in size, placement, and focus). Steps 5 and 6 deal with the second objective of improving the DNN (e.g., to better classify images when faced with rainy conditions). After briefly describing the CIFAR-10 benchmark and a DNN trained for it, each step in the data flow diagram is described in detail.

A) The CIFAR-10 Benchmark

The CIFAR-10 benchmark is commonly-used in research to assess a DNN's ability to perform image recognition [10]. The benchmark includes two datasets: a set of 50,000 labeled training images and a set of 10,000 labeled test images. The goal is to classify the contents of each image into one of ten categories: *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, or *truck*. State-of-the-art DNNs have been reported to achieve test accuracy above 95% on the default test data by use of highly-tuned architectures, optimization methods, and data augmentation [41] [42].

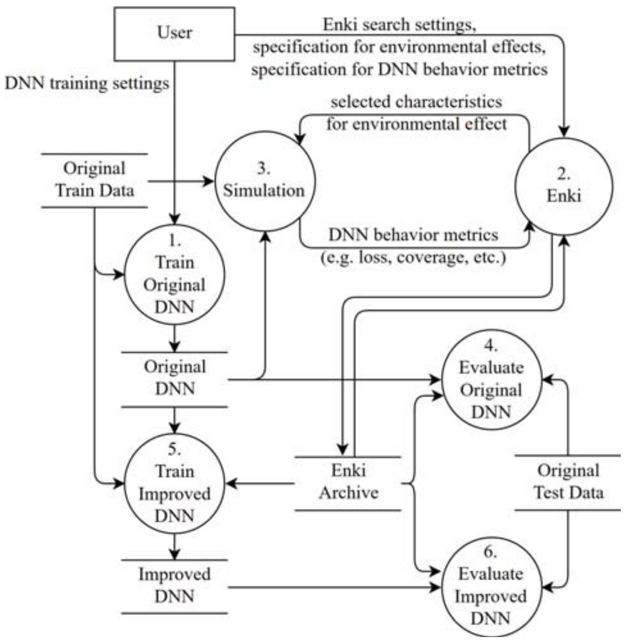

Fig. 2: Data flow diagram for assessing and improving a DNN for uncertain environmental conditions with Enki.

B) Assessing DNN Performance When Exposed to Uncertainty

For the current study, we require a pre-existing trained DNN as a basis for assessment. Here, a DNN has been constructed with the commonly-used ResNet [43] architecture, implemented with 20 activation layers. This architecture, shown in Fig. 3, is composed of a series of residual blocks with decreasing resolutions and increasing numbers of filters and "shortcut connections" to make each block optional during computation. Each block includes convolutional layers to transform incoming images, batch normalization operations to reduce covariant shift [44], and rectified linear unit (ReLU) [45] activations to filter out or amplify relevant features. Source images begin with a resolution of 32×32, pixels, and intermediate representations are gradually reduced in size down to a 8×8 resolution. The network then feeds the resulting features into a fully-connected layer, followed by a softmax[2] operation to predict the source image's category of classification (e.g., the *deer* category has the highest probability in the example provided in Fig. 3).

Step 1*) Train Original DNN:* A base-line DNN (M_{ORIG}) is trained on the original, unaltered CIFAR-10 dataset by means of an Adaptive Moment Estimation (Adam) gradient descent method [46] with a learning rate scheduler that decays the learning rate from 10^{-3} to 10^{-7} over 200 epochs.[3] (These

[2]Softmax functions are used to produce a probability distribution for the likelihood of data belonging to each possible classification category.

[3]An epoch involves iterating through each item in the training data.

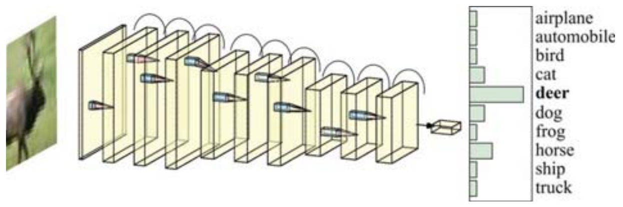

Fig. 3: High-level illustration of a CIFAR-10 DNN with a ResNet architecture. Color images are provided as input. The network is composed of a series of residual blocks with bypassing "shortcut connections." The final output is a probability distribution for predicting a classification category.

settings were chosen by empirical analysis.) Standard data augmentation methods such as image shifting and image flipping are used to add variation to the training data. Under these conditions, M_{ORIG} can be trained to achieve a test accuracy of 91% on the original test data (T_{ORIG}) provided by the CIFAR-10 benchmark.

***Step 2**) Enki:* Enki is responsible for discovering the impact of environmental uncertainty on a target system (e.g., M_{ORIG}). In this example, the goal is to discover different appearances of raindrops that, as a collection, produce unique and extreme behavior when processed by M_{ORIG}. Fig. 4 depicts the data flow within the Enki process.

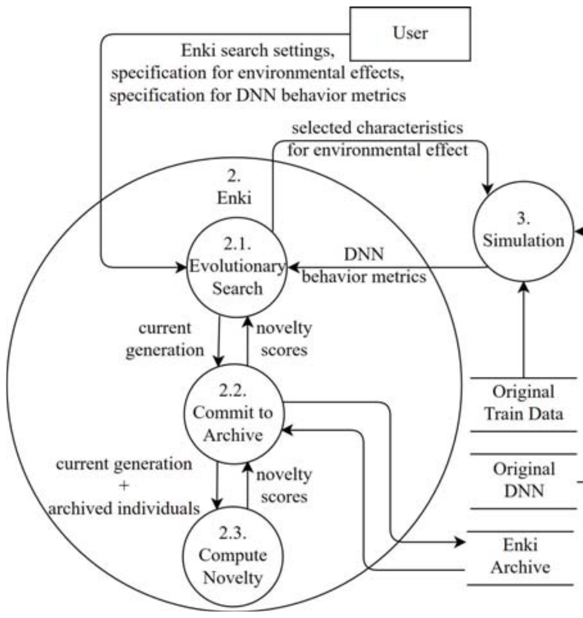

Fig. 4: Data flow diagram for the Enki process.

Enki implements an evolution-based search process (**Step 2.1**), described by the algorithm in Fig. 5. Enki evolves a population of individuals, each of which represents a different selected environmental effect and corresponds to a separate simulation (**Step 3**). Individuals are described by a genotype and phenotype that are specified by the user. The genotype encodes the characteristic properties of the environmental effect of interest (e.g., the position, radius, and amount of focus for a raindrop), and the phenotype encodes a set of behavior metrics observed when evaluating inputs under the corresponding environmental effect. (See **Step 3** for a description of the specific behavior metrics considered for this study.) Enki uses evolution to modify genomes within a population to encourage diversity with respect to the population's phenomes.

Algorithm 2 Enki

```
1: function EVOLUTIONARY-SEARCH($simulation$)
2:     $archive \leftarrow$ CREATE-ARCHIVE()
3:     $pop \leftarrow$ CREATE-RANDOM-POPULATION()
4:     $pop \leftarrow$ EVALUATE-BEHAVIOR($pop, simulation$)
5:     $archive, pop \leftarrow$ COMMIT-TO-ARCHIVE($archive, pop$)
6:     for all $generations$ do
7:         $pop \leftarrow$ SELECT($pop$)
8:         $pop \leftarrow$ RECOMBINE($pop$)
9:         $pop \leftarrow$ MUTATE($pop$)
10:        $pop \leftarrow$ EVALUATE-BEHAVIOR($pop, simulation$)
11:        $archive, pop \leftarrow$ COMMIT-TO-ARCHIVE($archive, pop$)
12:    end for
13:    return $archive$
14: end function
15: function COMMIT-TO-ARCHIVE($archive, pop$)
16:    $scores \leftarrow$ COMPUTE-NOVELTY($archive \cup pop$)
17:    $selection \leftarrow pop : scores > threshold$
18:    $archive \leftarrow$ TRUNCATE($archive \cup selection$)
19:    return $archive, pop$
20: end function
```

Fig. 5: Algorithm for the core SEARCH and ARCHIVE functions of Enki.

An initial population is produced by generating a set of random genomes, and each consecutive generation is evolved by performing a series of selection, crossover, and mutation operations on the genomes. Enki also maintains an archive (**Step 2.2**) that includes the most diverse individuals from both past and present generations. With each new generation, Enki selects which individuals to reproduce by comparing their *novelty scores* via tournament selection. Novelty scores are computed for each individual (**Step 2.3**) by calculating the average Euclidean distance between each corresponding behavior metric in an individual's phenome with its nearest neighbors in the archive. After individuals are selected, a single-point crossover operation is used to recombine their genomes, which is then followed by a mutation operator that shifts each encoded value in the genome according to a mutation rate. The population is then committed to the archive, where novelty scores are recomputed with the individuals currently in the archive, and the archive is updated to include individuals with the highest novelty scores. The archive is implemented with a fixed size, such that individuals with the lowest novelty scores are discarded. When the mean novelty score for individuals in the archive converges between generations, Enki is terminated and the resulting archive contains a set of environmental characteristics (e.g., different appearances of raindrops) that have been observed to cause distinctly different behavior in the target system.

Step 3) *Simulation:* None of the original images in the CIFAR-10 benchmark were taken in the presence of visible rain. In order to assess the DNN under rainy conditions, raindrops are simulated via a ray tracing technique. Garg and Nayar [47] have written extensively about modeling the visual appearance of rain to a high degree of detail. However, these models are often complex and computationally expensive when applied to datasets as large as those normally required for training or evaluating a DNN. Raindrops falling onto a camera lens can either settle as stationary pools of water or even produce streaks, depending on factors such as the angle of gravity, lens curvature, and wind. For simplification, this paper assumes raindrops are hemispherical and stationary on the lens. These simulated raindrops can be described by the following properties: position, radius, and blur. For comparison, Fig. 6 shows example images taken with real-world water droplets on a camera lens (Fig. 6(b)) next to the simulated effects (Fig. 6(c)), applied to images in the absence of water droplets (Fig. 6(a)). Since the appearance of real-world raindrops include rays of light coming from sources outside of the image, it is not possible for these simulated raindrops to completely match the real-world examples, but it is posited that the reality gap is sufficiently small enough to ignore for this case study.

To assess the effect of raindrop occlusion on M_{ORIG}, the simulation procedure uses a set of 1,000 randomly selected CIFAR-10 training images. Each image is transformed with the same raindrop effect, defined by its position, radius, and blur. After constructing the synthetic images, the simulation then evaluates the entire collection of synthetic images with M_{ORIG} and three behavior metrics are observed: the DNN's classification *error*, activation *coverage*, and activation *pattern*. Fig. 7 shows an algorithm for the simulation procedure.

The classification error (i.e., $E(\mathbf{x})$) of a DNN is defined by the categorical cross entropy function

$$E(\mathbf{x}) = \frac{1}{n}\sum_{i=0}^{n} t_i \log y_i \quad (1)$$

where $\mathbf{x}$ is an input image for the DNN, $\mathbf{t} = \langle t_1, t_2, \ldots, t_n\rangle$ is the ground truth classification label associated with $\mathbf{x}$, represented by a one-hot encoding[4] of the ground truth classification, and $\mathbf{y} = \langle y_1, y_2, \ldots, y_n\rangle$ is the DNN's predicted classification label for $\mathbf{x}$. When training the DNN, the goal is to minimize this function. Sets of synthetic images resulting in lower error values indicate that a DNN is more accurate with its predictions. By using Enki to diversify individuals based on this metric, the aim is to find environmental effects that result in a broad range of accuracy for the DNN under test.

A DNN's activation coverage (i.e., $C(\mathbf{x})$) is computed by monitoring the output of each neuron in each activation layer of the DNN and computing the ratio of activated neurons to the entire set of neurons. Thus, the neuron coverage indicates the fraction of the network exercised by the set of given synthetic images. Since each neuron is activated with a ReLU function, it is considered activated when it has any value greater than zero, and therefore, the activation coverage can be computed

[4]A one-hot encoding is a binary vector with dimensions equal to the number of possible classes. Only the element corresponding to the matching class is designated with a 1.

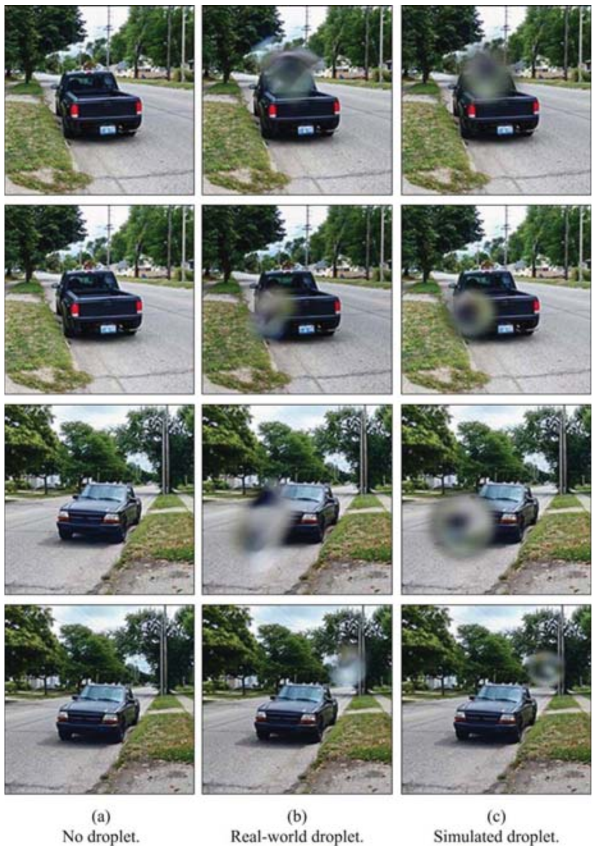

Fig. 6: Examples of real-world and simulated water droplets. Column (a) contains an absence of any droplets, (b) contains real-world droplets, and (c) contains simulated droplets.

Algorithm 1 Simulation Procedure

```
1: function SIMULATE($model, train\_subset, params$)
2:     $synth\_subset \leftarrow$ TRANSFORM($train\_subset, params$)
3:     $error \leftarrow$ EVALUATE($model, synth\_subset$)
4:     $coverage \leftarrow$ COMPUTE-COVERAGE($model, synth\_subset$)
5:     $pattern \leftarrow$ COMPUTE-PATTERN($model, synth\_subset$)
6:     return $loss, coverage, pattern$
7: end function
```

Fig. 7: Algorithm for evaluating a DNN in the presence of a simulated environmental condition.

as follows:

$$C(\mathbf{x}) = \frac{\sum_{\ell \in L} \sum_{h \in \ell} \sigma'(h)}{\sum_{\ell \in L} |\ell|} \tag{2}$$

$$\sigma'(h) = \begin{cases} 1 & \sigma(h) > 0 \\ 0 & \sigma(h) = 0 \end{cases} \tag{3}$$

where L is the set of all activation layers in the DNN, ℓ is the set of neurons in an activation layer, h is an individual neuron, and $\sigma(h)$ is the activation value for a neuron when the DNN is given input image $\mathbf{x}$. By including this metric, the aim is to discover a set of environmental effects that cover a spectrum ranging from minimal execution of the DNN under test to maximum.

The activation pattern (i.e., $P(\mathbf{x})$) is computed similarly to its activation coverage. A single vector is constructed with elements having a one-to-one correspondence to all of the neurons in each activation layer of the DNN. Elements corresponding to activated neurons are assigned a value of 1, whereas all others are assigned a value of 0.

$$P(\mathbf{x}) = \langle \sigma'(h) | \forall h \in \ell, \forall \ell \in L \rangle \tag{4}$$

Fig. 8 illustrates two examples of possible activation patterns for the same simple neural network. Activated neurons, depicted with check marks, contribute to the final output, while non-activated neurons have zero contribution. These patterns are observed as a means for measuring which portions of the DNN are exercised by a specific set of images, analogous to the execution paths for a traditional software program. Including this metric for Enki to diversity will force each environmental effect in the archive to activate mutually unique patterns of neurons in the DNN under test.

Step 4*) Evaluate Original DNN:* The resulting archive from Enki contains a set of environmental effects that can be used to construct new, synthetic test images to assess the DNN's performance. Each individual environmental effect in the archive has unique values for the *transformation parameters* associated with it. In this example, the transformation function is the procedure used to generate the raindrop effect, and the transformation parameters are the *position* of the raindrop, *radius* of the raindrop on the image, and the *blur* factor within the raindrop. A synthetic test dataset (T_{ENKI}) can be produced by taking the unaltered CIFAR-10 test dataset (T_{ORIG}) and applying the transformation function with the transformation parameters associated with each individual in Enki's archive. Note that, prior to this step, images in T_{ORIG} are deliberately not used to inform which transformation properties should be applied. Thus, T_{ENKI} has not been contaminated with any information that could be inferred from T_{ORIG} and bias the performance of M_{ORIG} more favorably towards it.

In this step, M_{ORIG} is evaluated against both T_{ORIG} and T_{ENKI}, in order to compare the performance of M_{ORIG} in the absence and presence of the uncertain environmental condition. Performance is determined by observing the overall accuracy of M_{ORIG} when predicting image classification categories.

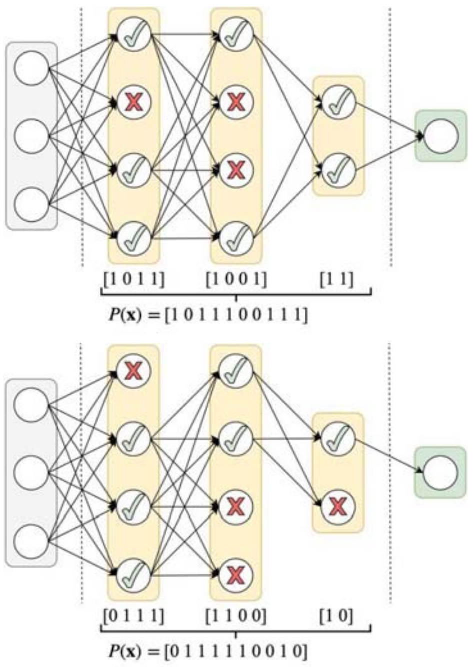

Fig. 8: Example activation patterns for the same DNN. When computing $P(\mathbf{x})$, activated neurons (marked with checks), are assigned a value of 1, and inactive neurons are assigned a value of 0.

C) Improving the DNN For Exposure to Uncertainty

After assessing the impact of an uncertain environmental condition on M_{ORIG}, the second objective is to improve the performance of the DNN in the context of the same environmental condition.

Step 6*) Train Improved DNN:* Synthetic training data is generated by applying the transformation function to the the original CIFAR-10 training data with the transformation parameters found by Enki in **Step 2**. A new DNN (M_{ENKI}) is then created by using a combination of the original CIFAR-10 training data and the synthetic training data. M_{ENKI} is initialized with the weights from M_{ORIG} and trained for 100 epochs in a procedure similar to **Step 1** but with an initial learning rate set to 10^{-7} to resume training at the same rate where the training phase ended with M_{ORIG}.

Step 7*) Evaluate Improved DNN:* Upon completion of training, M_{ENKI} can be evaluated against both test datasets T_{ORIG} and T_{ENKI}, and results can be compared to those found for M_{ORIG} in **Step 4**.

IV. Experiments and Results

This section describes in detail the experiments conducted with Enki on a CIFAR-10 DNN under a combination of uncertain environmental conditions. Section III described how a single raindrop condition could be applied with Enki. However, additional environmental conditions, such as variable brightness and contrast, have been included in these experiments. For validation and comparison to alternative approaches, our experiments have also been replicated with the DeepTest method and a random generation method.[5] For brevity, test datasets generated by Enki, DeepTest, and random generation will be labeled T_{ENKI}, T_{DEEP}, and T_{RAND} respectively. Likewise, new DNNs trained with training data from each method will be respectively labeled M_{ENKI}, M_{DEEP}, and M_{RAND}.

A. Experiment Setup

Each experiment executes the process described in Section III and Fig. 2. Table I defines the transformation parameters and their value ranges. A *permutation* parameter is included to enable Enki to apply any permutation of one or more environmental conditions (e.g., variable brightness, variable contrast, and/or a raindrop on the lens). Enki is executed with the configuration in Table II, resulting in an archive of 50 different transformation functions associated with the selected environmental effects. These transformation functions are then applied to the entire CIFAR-10 test dataset to create T_{ENKI}. A new DNN is trained with a combination of the unaltered CIFAR-10 training data and synthetic training data from Enki's transformation functions to create M_{ENKI}.

Additionally, each experiment executes the DeepTest algorithm with the same possible transformation functions and transformation parameters to create a synthetic test dataset T_{DEEP} and DNN M_{DEEP}. A random method is also executed to apply any random permutation of the transformation functions to each training and test image, with random values for the transformation parameters, to create T_{RAND} and M_{RAND}.

TABLE I: Transformation Parameters

Parameter	Value Range
permutation	all permutations of transform. funcs.
brightness	0% to 100%
contrast	0% to 100%
raindrop_x	0 to 31 pixels
raindrop_y	0 to 31 pixels
raindrop_radius	0 to 10 pixels
raindrop_blur	1.0 to 2.0 pixels

TABLE II: Enki Configuration

Setting	Value
num_generations	50 generations
archive_size	50 individuals
population_size	10 individuals
tournament_size	3 comparisons
mutation_rate	14%
mutation_shift	20%
num_nearest_neighbors	3 individuals

B. Assessing the Impact of Environmental Uncertainty

Fig. 9 shows the results from evaluating the original CIFAR-10 DNN (M_{ORIG}) on test data from each method. When diversifying environmental effects by the neuron coverage and loss produced by each effect, Enki was able to produce a test dataset (T_{ENKI}) that reduced the accuracy of M_{ORIG} from 91% to a mean of 21% over ten separate trials, whereas T_{DEEP} and T_{RAND} only reduced the accuracy to a mean of 52% and 60% respectively. Standard deviations (σ) are also included in Fig. 9. This result shows that Enki was able to discover specific environmental effects that had a greater adverse impact on the DNN (i.e., gaps in the training data). Furthermore, since Enki's environmental effects are uncoupled from a specific synthetic image, the transformation parameters associated with each effect can be examined to give insight on which types of environmental effects are involved in the adverse cases discovered. With this information, developers can target specific strategies to mitigate the gaps (e.g., use alternate sensors, supplement sensors, etc.). Fig. 10 shows a heat map of the values of each transformation parameter found in Enki's archive. Regions colored more yellow indicate a larger presence of environmental effects with that value in the archive. From this information, it can be seen that Enki was automatically able to discover that darker, low-contrast images with raindrops near the center caused the most impact on M_{ORIG}.

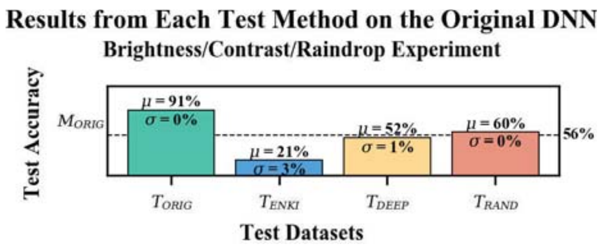

Fig. 9: Accuracy of the original DNN when applied to each test dataset. Bars are annotated with the mean accuracy (μ) and standard deviation (σ) over ten separate trials. The overall mean accuracy is indicated by the dashed line.

C. Improving Performance under Environmental Uncertainty

Fig. 11 shows the results from evaluating each DNN trained with additional synthetic data from each method on each test dataset. The overall mean accuracy for each DNN is indicated by the dashed line. Each enhanced DNN has shown improvement across all test datasets, verifying that synthetic training data can be useful for making the DNN more robust. Comparable results where found for each DNN when evaluated against T_{ORIG}, T_{DEEP}, and T_{RAND}. However, the accuracy observed by each DNN on T_{ENKI} remains significantly lower, with M_{ENKI} performing the best against it. This result indicates that Enki was able to find significantly more adverse environmental effects than the other methods, and furthermore, it was able to create a DNN more capable of handling those effects.

[5]DeepXplore requires multiple DNNs and is therefore not comparable to our approach.

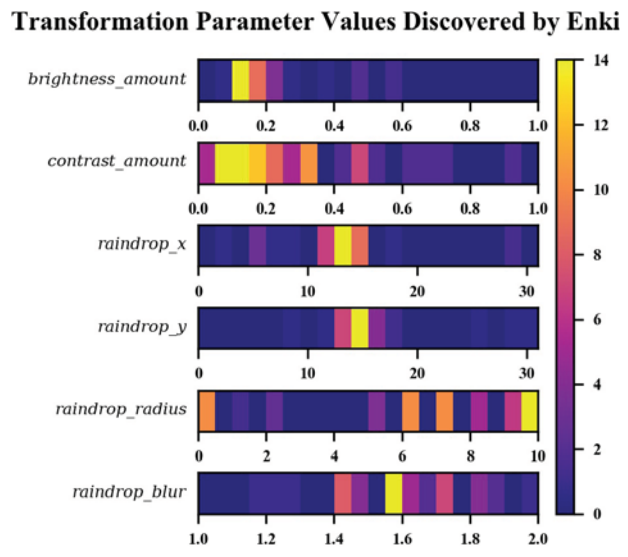

Fig. 10: Heat map visualization of values discovered by Enki for each transformation parameter. Regions colored more yellow have more corresponding values covered.

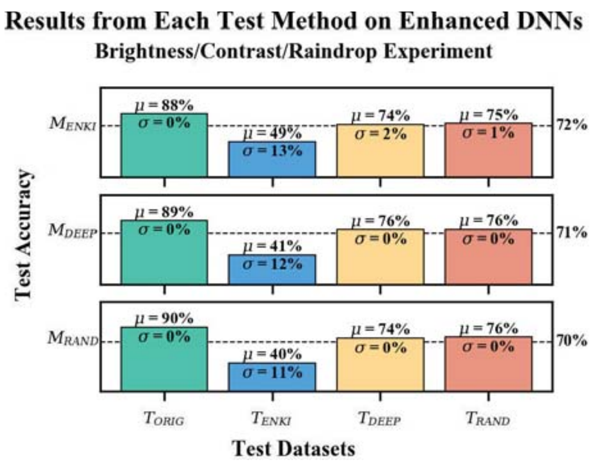

Fig. 11: Accuracy of DNNs enhanced with synthetic training data from each method when applied to each test dataset. Bars are annotated with the mean accuracy (μ) and standard deviation (σ) over ten separate trials. The overall mean accuracy for each DNN is indicated by the dashed line.

D. Threats to Validity

The results in this paper are limited to a DNN with the ResNet architecture and trained for the CIFAR-10 dataset. The effectiveness of Enki, DeepTest, or a random approach on alternative architectures and datasets will likely vary. Additionally, these techniques all contain stochastic elements. Multiple trials have been conducted for the purpose of this paper to show how much variance may be expected in the results for each technique. This work has also been limited to evaluating a DNN against simulated environmental effects, and results have not been verified against real-world analogues. Future research will explore the effectiveness of Enki on different DNN architectures, problem spaces, and interactions with real-world data.

V. Conclusion

This paper has introduced the use of novelty search to augment data for an LES in order to assess and enhance its performance under new environmental conditions with uncertain effects. It has been demonstrated that Enki can be used to generate synthetic test data to assess a DNN under environmental conditions not covered by existing test data and generate synthetic training data to improve a DNN's overall accuracy under specified environmental conditions of uncertainty. Because Enki provides a set of transformation functions for environmental effects that are not tied to any specific input data, alternative applications can be considered. Future research will explore the use of ensemble methods with sub-classifiers that specialize in each individual environmental effect discovered by Enki. Additional research will be conducted with Enki to discover execution modes for cyber-physical SASs and alternative strategies for uncertain environmental conditions [48] [49], including onboard controllers [50].

Acknowledgments

We greatly appreciate the feedback from the anonymous ICAC reviewers, as well as detailed comments from Philip McKinley on early versions of this paper.

This work has been supported in part by grants from NSF (CNS-1305358 and DBI-0939454), Ford Motor Company, and General Motors Research; and the research has also been sponsored by Air Force Research Laboratory (AFRL) under agreements FA8750-16-2-0284 and FA8750-19-2-0002. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory (AFRL), the U.S. Government, National Science Foundation, Ford, GM, or other research sponsors.

References

[1] "Report of the Defense Science Board Summer Study on Autonomy," U.S. Department of Defense, Tech. Rep., 2016.
[2] C. E. Tuncali, J. Kapinski, H. Ito, and J. V. Deshmukh, "Reasoning About Safety of Learning-enabled Components in Autonomous Cyber-physical Systems," in *Proceedings of the 55th Annual Design Automation Conference (DAC)*, 2018, pp. 30:1–30:6.
[3] "Accident Report, NTSB/HAR-17/02, PB2017-102600," National Transportation Safety Board (NTSB), Tech. Rep., 2017.
[4] "Preliminary Report, Highway, HWY18FH011," National Transportation Safety Board (NTSB), Tech. Rep., 2018.
[5] "Preliminary Report, Highway, HWY18MH010," National Transportation Safety Board (NTSB), Tech. Rep., 2018.
[6] H. He and E. A. Garcia, "Learning from Imbalanced Data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, Sept 2009.
[7] V. Sessions and M. Valtorta, "The Effects of Data Quality on Machine Learning Algorithms," in *The International Conference on Information Quality (ICIQ)*, 2006.

[8] C. E. Brodley and M. A. Friedl, "Identifying Mislabeled Training Data," *Journal Of Artificial Intelligence Research*, vol. 11, 1999.

[9] G. Calikli and A. Bener, "Empirical Analyses of the Factors Affecting Confirmation Bias and the Effects of Confirmation Bias on Software Developer/Tester Performance," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering (PROMISE)*, 2010.

[10] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," 2016.

[11] J. Janai, F. GÜney, A. Behl, and A. Geiger, "Computer Vision for Autonomous Vehicles: Problems, Datasets and State-of-the-Art," *CoRR*, vol. abs/1704.05519, 2017.

[12] A. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep Reinforcement Learning Framework for Autonomous Driving," *Electronic Imaging*, 2017.

[13] O. Strömgren, "Deep Learning for Autonomous Collision Avoidance," MSc Thesis, Linköping University, 2018.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[15] W. Knight, "The Dark Secret at the Heart of AI," MIT Technological Review, April 2017. [Online]. Available: https://www.technologyreview.com/s/604087/the-dark-secret-at-the-heart-of-ai/

[16] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba, "Network Dissection: Quantifying Interpretability of Deep Visual Representations," *CoRR*, vol. abs/1704.05796, 2017.

[17] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding Neural Networks Through Deep Visualization," in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 2015.

[18] J. Schumann, P. Gupta, and Y. Liu, "Application of Neural Networks in High Assurance Systems: A Survey," in *Applications of Neural Networks in High Assurance Systems*, J. Schumann and Y. Liu, Eds. Springer, 2010, ch. 1, pp. 1–40.

[19] S. Burton, L. Gauerhof, and C. Heinzemann, "Making the Case for Safety of Machine Learning in Highly Automated Driving," in *International Conference on Computer Safety, Reliability, and Security*, S. Tonetta, E. Schoitsch, and F. Bitsch, Eds. Springer, 2017, vol. 10489.

[20] J. Ding, X. Kang, and X.-H. Hu, "Validating a Deep Learning Framework by Metamorphic Testing," in *Proceedings of the 2Nd International Workshop on Metamorphic Testing (MET)*, 2017.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS)*, vol. 1, 2012.

[22] S. R. Kheradpisheh, M. Ghodrati, M. Ganjtabesh, and T. Masquelier, "Deep Networks Can Resemble Human Feed-forward Vision in Invariant Object Recognition," *Scientific Reports*, vol. 6, no. 32672, 2016.

[23] I. J. Goodfellow, J. Shlen, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," in *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[24] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial Machine Learning at Scale," in *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

[25] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song, "Robust Physical-World Attacks on Machine Learning Models," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[26] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing Properties of Neural Networks," in *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2014.

[27] A. Nguyen, J. Yosinski, and J. Clune, "Deep Neural Networks Are Easily Fooled: High Confidence Predictions For Unrecognizable Images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[28] J. Jo and Y. Bengio, "Measuring the Tendency of CNNs to Learn Surface Statistical Regularities," *CoRR*, vol. abs/1711.11561, 2017.

[29] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based Software Engineering: Trends, Techniques and Applications," *ACM Comput. Surv.*, vol. 45, no. 1, 2012.

[30] P. McMinn, "Search-Based Software Testing: Past, Present and Future," in *Proceedings of the 4th International Conference on Software Testing, Verification and Validation Workshops (ICST)*, 2011.

[31] G. Fraser and A. Arcuri, "EvoSuite: Automatic Test Suite Generation for Object-oriented Software," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE)*, 2011.

[32] K. Mao, M. Harman, and Y. Jia, "Sapienz: Multi-objective Automated Testing for Android Applications," in *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA)*, 2016.

[33] J. Lehman, "Evolution Through the Search for Novelty," Ph.D. dissertation, University of Central Florida, 2012.

[34] J. Lehman and K. Stanley, "Novelty Search and the Problem with Objectives," *Genetic Programming Theory and Practice IX*, 2011.

[35] A. J. Ramirez, A. C. Jensen, B. H. C. Cheng, and D. B. Knoester, "Automatically Exploring How Uncertainty Impacts Behavior of Dynamically Adaptive Systems," in *Proceeings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2011, pp. 568–571.

[36] F. Dalpiaz, A. Borgida, J. Horkoff, and J. Mylopoulos, "Runtime Goal Models: Keynote," in *Proceedings of 7th IEEE International Conference on Research Challenges in Information Science (RCIS)*, 2013.

[37] M. Boussaa, O. Barais, G. Sunye, and B. Baudry, "A Novelty Search-based Test Data Generator for Object-oriented Programs," in *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO)*, 2015.

[38] M. Boussaa, O. Barais, G. Sunyé, and B. Baudry, "A Novelty Search Approach for Automatic Test Data Generation," in *Proceedings of the 8th International Workshop on Search-Based Software Testing (SBST)*, 2015.

[39] K. Pei, Y. Cao, J. Yeng, and S. Jana, "DeepXplore: Automated Whitebox Testing of Deep Learning Systems," in *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, 2017.

[40] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2018.

[41] S. Zagoruyko and N. Komodakis, "Wide Residual Networks," in *Proceedings of the British Machine Vision Conference (BMVC)*, 2016.

[42] X. Gastaldi, "Shake-Shake Regularization of 3-branch Residual Networks," in *In Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016, pp. 770–778.

[44] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015.

[45] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML)*, 2010.

[46] D. Kingma and J. Ba, "Adam: a Method for Stochastic Optimization," in *Proceedings for the International Conference on Learning Representations (ICLR)*, 2015.

[47] K. Garg and S. K. Nayar, "Photometric Model of a Rain Drop," Columbia University, Tech. report, 2004. [Online]. Available: http://www1.cs.columbia.edu/CAVE/publications/pdfs/Garg_TR04.pdf

[48] A. J. Clark, B. DeVries, J. M. Moore, B. H. C. Cheng, and P. K. McKinley, "An Evolutionary Approach to Discovering Execution Mode Boundaries for Adaptive Controllers," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016.

[49] C. Richter and N. Roy, "Safe Visual Navigation via Deep Learning and Novelty Detection," in *Robotics: Science and Systems XIII, Massachusetts Institute of Technology*, 2017.

[50] M. A. Langford, G. A. Simon, P. K. McKinley, and B. H. C. Cheng, "Applying Evolution and Novelty Search to Enhance the Resilience of Autonomous Systems," in *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2019.