VISION: Evaluating Scenario Suitableness for DNN Models by Mirror Synthesis

Ziqi Chen, Huiyan Wang, Chang Xu, Xiaoxing Ma, Chun Cao
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
Department of Computer Science and Technology, Nanjing University, Nanjing, China
rubychen0611@yeah.net, cocowhy1013@gmail.com, {changxu, xxm, caochun}@nju.edu.cn

***Abstract*—Software systems assisted with deep neural networks (DNNs) are gaining increasing popularities. However, one outstanding problem is to judge whether a given application scenario suits a DNN model, whose answer highly affects its concerned system's performance. Existing work indirectly addressed this problem by seeking for higher test coverage or generating adversarial inputs. One pioneering work is SynEva, which exactly addressed this problem by synthesizing mirror programs for scenario suitableness evaluation of general machine learning programs, but fell short in supporting DNN models. In this paper, we propose VISION to eValuatIng Scenario suItableness fOr DNN models, specially catered for DNN characteristics. We conducted experiments on a real-world self-driving dataset Udacity, and the results show that VISION was effective in evaluating scenario suitableness for DNN models with an accuracy of 75.6–89.0% as compared to that of SynEva, 50.0–81.8%. We also explored different meta-models in VISION, and found out that the decision tree logic learner meta-model could be the best one for balancing VISION's effectiveness and efficiency.**

***Index Terms*—DNN model, mirror synthesis, scenario suitableness evaluation**

I. Introduction

There is an increasing trend of deploying machine learning (ML) programs with their trained models (e.g., Deep Neural Network or DNN models) for solving practical problems, e.g., self-driving [1], medical diagnostics [2], image processing [3], and machine translation [4]. Unlike traditional programs, which can be verified against their specifications, ML programs are difficult to tell whether their trained models *suit* a specific application scenario. It is more challenging for DNN programs due to lacking an interpretation to their trained DNN models, which consist of massive neurons. As a result, a DNN program can cause abnormal behaviors or even disasters due to its low accuracy if deployed to an application scenario unsuitable for its trained model. For example, according to a survey on 5,328 disengagements of autonomous vehicles [5], as high as 64% cases were found to be buggy in ML systems (mostly DNN-based), among which the low accuracy of image classification was the dominant cause.

The above problem relates to the evaluation on the suitableness of a trained DNN model against an application scenario. Traditional practices for testing trained DNN models rely on manual labeling and inspection [6], [7], which is very time- and effort-consuming, infeasible in practice. Some recent work transformed this problem into a test adequacy one by measuring structural test coverage criteria [8], [9], [10], but a model with higher coverage does not imply more suitableness [11]. Some other work tried to diagnose a DNN model by generating adversarial inputs that can cause the model to predict wrongly, by means of metamorphic testing [12], [13], differential testing [8], [14] or distance measurement [15], [16], but these attempts did not directly answer the suitableness question.

One pioneering work, SynEva [17], proposed to evaluate the suitableness of a trained ML model against a given application scenario. It constructs a mirror model from the original model, which guarantees to behave similarly as the original model for a scenario that is sufficiently similar to the training one from which the model is instantiated, but could behave differently from the original model for a scenario that deviates significantly from the training one. By doing so, SynEva distinguishes these two types of scenarios.

SynEva works well, and is not restricted to certain ML algorithms (even applicable to the k-means algorithm [18], which was not originally for classification). However, SynEva has never been explicitly applied to DNN models and its effectiveness is unclear. As such, we investigated its application to DNN models and identified the following problems due to its inherent limitations: (1) a trained DNN model usually consists of massive neurons, which can cause SynEva to be overwhelmed by huge time and space overheads since it has to construct logic learners for all neurons in mirroring (e.g., a 10-layer CNN's mirroring time can easily last for months); (2) SynEva's performance could also be restricted due to its support vector machine (SVM) choice as the only logic learner (e.g., only 50.0% accuracy in the worst case); (3) SynEva relies essentially on control flows in its synthesized mirror model, but DNN models work in a data-flow way [19], so that its inherent path-based similarity measurement cannot effectively distinguish different scenarios.

In this paper, we propose VISION for eValuatIng Scenario suItableness fOr DNN models. It follows SynEva's key mirror synthesis idea but carefully addresses the aforementioned limitations. In general, VISION consists of three working phases, namely, *model preparation* (traditional training), *mirror synthesis*, and *suitableness measurement*, the latter two of which are VISION's main parts and together address the analyzed three problems. First, only neurons from partial layers (e.g., fully connected layers) are considered in VISION for logging concise information and synthesizing logic learners in the mirroring, and this treatment addresses the *overhead* problem.

Second, Vision integrates multiple selections of logic learner meta-model (including three linear and three non-linear ones), and this treatment makes Vision customizable and addresses the *choice* problem. Third, instead of relying on control flows, Vision works in a data flow way by taking neurons' output values into consideration for synthesizing mirror models and measuring their suitableness based on distances, and this treatment addresses the *flow* problem.

We experimentally evaluated our Vision on a practical self-driving dataset released by Udacity [20] with DNN model DAVE-2 [1]. The experimental results show that when both integrated with the SVM logic learner meta-model, Vision was effective in distinguishing suitable and unsuitable application scenarios with an accuracy of 75.6–89.0% (mean: 82.3%), while SynEva behaved with an accuracy of 50.0–81.8% (mean: 69.8%). We also tested Vision with different logic learner meta-models, and observed that its effectiveness varied and when integrated with the k-nearest neighbors meta-model it achieved the best accuracy of 88.0–93.8% (mean: 91.1%). Besides, we measured Vision's time and space overheads, and observed that the overheads varied with different logic learner meta-models, and the decision tree one was the most preferred that best balanced the effectiveness and efficiency.

The key contributions of this paper are as follows:

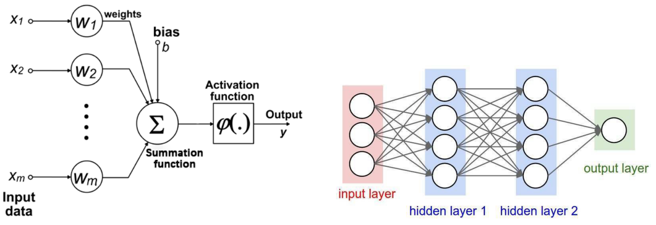
- Proposal of the Vision approach to automatically evaluating scenario suitableness for DNN models.
- Evaluation of Vision on a real-world dataset, whose results confirmed its effectiveness (accuracy) in practice.
- Measurement of time and space overheads of Vision for suggestions in its practical usage.

The rest of this paper is organized as follows. Section II introduces the DNN background. Section III gives Vision's overview and elaborates on its main working phases. Section IV experimentally evaluates Vision's performance and compares it to existing work. Section V presents related work in recent years, and finally Section VI concludes this paper.

II. Background

DNN, as one type of Artificial Neural Network (ANN), is composed of *neurons*, following the M-P Neuron Model, as shown in Fig. 1(a). In this model, a neuron receives multiple inputs from n other neurons, which are transmitted through weighted connections. The total input value $\sum_{i=1}^{n} w_i x_i$ received by the neuron is combined with a threshold value b associated with this neuron, and then processed by a summation function $\sum_{i=1}^{n} w_i x_i - b$ and activation function φ to produce the output of the neuron, i.e., $y = \varphi(\sum_{i=1}^{n} w_i x_i - b)$. When multiple neurons are connected in a hierarchy, a neural network is formed. In general, a neural network can be considered as a mathematical model with lots of parameters, which aggregates effects of its nested functions.

To be specific, a DNN consists of an *input layer* and an *output layer*, as well as multiple *hidden layers* in between, as shown in Fig. 1(b). Hidden layers progressively extract higher-level features from raw inputs to discover distributed feature representations of input data. A conventional DNN is typically fully connected, i.e., neurons of two adjacent layers are fully connected to each other. More diverse DNN structures have also been proposed for solving specific problems. For example, Convolutional Neural Network (CNN) is a wide-used DNN architecture, which has brought about breakthroughs in processing images, speeches, and videos. The hidden layers of a CNN typically consist of a series of convolutional layers, pooling layers, and fully connected layers. A neuron in a convlutional layer connects only to partial neurons in its last layer, and its computational process can be represented as a convolution with kernels. Pooling layers are usually responsible for dimension reduction.



(a) M-P neuron model [21]

(b) Basic architecture of a DNN [22]

Fig. 1. M-P neuron model and DNN architecture.

III. Approach

In this section, we elaborate on our Vision approach, and explain how it evaluates scenario suitableness for DNN models, while addressing SynEva's limitations.

A. Overview

Fig. 2 gives Vision's overview, which consists of three working phases. Phase 1 (*model preparation*) trains a DNN model based on training instances from a given training scenario according to some pre-selected deep learning (DL) algorithm, and refers to this trained model by the knowledge model (or *knowledge*). Then, Phase 2 (*mirror synthesis*) synthesizes a mirror model (or *mirror*) that behaves similarly to the knowledge model upon training instances, by replacing the logics of the neurons in the knowledge model with newly trained logic learners. Finally, Phase 3 (*suitableness measurement*) measures the *behavioral differences* (BD) for instances in a given new scenario to evaluate whether the knowledge and mirror models still behave similarly to each other, and thus decides whether the tested instances or the whole scenario (all instances) suit(s) the knowledge model (or the previously DNN model), according to Vision's internal adaptive BD threshold learner.

Among the three phases, Phase 1 is the traditional DNN training. Therefore, we further elaborate on mirror synthesis and suitableness measurement in the following.

B. Mirror Synthesis

In this phase, Vision synthesizes a mirror model M that behaves similarly in the prediction on the training scenario with the trained knowledge model K in Phase 1, and uses the difference between the two models (knowledge and mirror) as

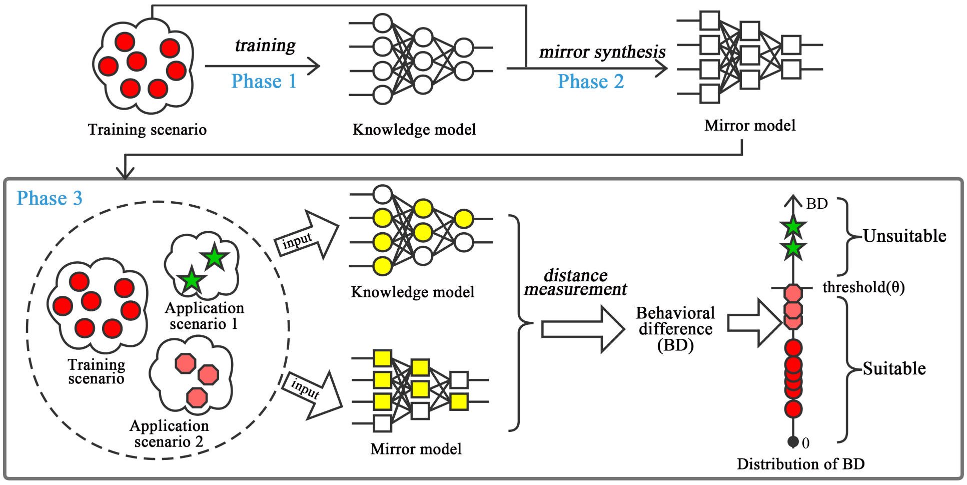


Fig. 2. VISION Overview

the pseudo-oracle in Phase 3 to distinguish suitable (when K and M behave similarly) and unsuitable (otherwise) scenarios.

For synthesizing such a mirror model, VISION follows SynEva's idea of replicating the whole structure of the knowledge model K to the mirror model M, but replacing M's neuron logics with newly trained logic learners. To be specific, VISION conducts the following two steps: (1) synthesizing M by a DNN-style model with the same structure as K (including its neuron layers, neuron number of each layer, weight values, and bias values); (2) constructing M's neuron logics by new logic learners, which make M behave similarly with K on the original training scenario. The first step is straightforward by structure copying, but the second step is more complicated. For replacing M's neuron logics, which neurons to be replaced for new logics (the *which* problem) and how to replace these logics with new learners should be answered (the *how* problem), and we explain them below.

For the which problem, considering SynEva's overhead problem about coping with massive neurons in a DNN model, VISION selects only neurons from fully connected (FC) layers for their logic replacement. The reason is that FC layers tend to inherit more functional mechanisms [23], while other layers like convolutional or pooling ones tend to extract or reduce dimensional features. This treatment can realize a better trade-off for balancing the effectiveness and efficiency.

For the how problem, VISION considers to train a new logic learner for each selected neuron based on necessary logged information (including each neuron's input matrix and summation value), so as to simulate the mirror model M's behavior as the knowledge model K does. To be specific, VISION feeds all training instances from the training scenario into knowledge K and logs relevant information for the selected neurons. VISION then uses the logged information to train new logic learners as *regressors* by a selected regression meta-model algorithm, i.e., taking neuron inputs as features, and its summation values as labels. In this way, each neuron is associated with a learner that is responsible for this neuron to generate the output of its summation function. Then this neuron's activation function generates its follow-up output value as knowledge K does.

Recall that SynEva has a similar logic learner training process. It trains each learner as a *classifier* by the SVM classification algorithm to decide whether to activate one or more follow-up edges as its output in the synthesized mirror model, assuming that its model execution in the prediction mostly works in a control-flow way. However, DNN models essentially work in a data-flow way in the prediction process [19]. As such, the output values of each neuron are valuable for simulating knowledge K's behavior, neglecting which would lead to poor performance when SynEva is applied to DNN models. Regarding this, VISION trains each learner to be a regressor as aforementioned (addressing SynEva's flow problem). Moreover, to cope with SynEva's choice problem about its only SVM meta-model choice, VISION extends to integrate more logic learner meta-models to be customizable. VISION considers six common regression algorithms for meta-model choices, including three linear ones: ridge regression (*Ridge*), least squares regression (*LinearRegression*) and linear support vector regression (*LinearSVR*), and three non-linear ones: support vector regression (*SVR*), decision tree regression (*DT*) and k-nearest neighbors regression (*KNN*).

Formally, the logic learner training process for mirror synthesis phase is presented in Algorithm 1. Suppose that there are k FC layers in knowledge K VISION considers in mirroring, and the i^{th} layer (1 to k) has t_i neurons. Suppose that the training scenario contains m instances. Let the j^{th} neuron at the i^{th} layer have weight $\mathbf{w}_{ij}$ (t_{i-1}-dimensional vector) and bias b_{ij}. Then, for any neuron at the i^{th} layer, it receives inputs from t_{i-1} neurons at its last layer, and the m instances' inputs together form the input matrix $\mathbf{X}_i$ (size of $m \times t_{i-1}$). Next, the outputs of summation function $\mathbf{y}_{ij}$ are calculated from corresponding weight and bias values at Line 3, where $\mathbf{e}$ is a m-dimensional vector with all ones. Based on a specific selected meta-model, a logic learner is initialized (Line 4) and trained for all instances' inputs in the matrix (Line 5) by feeding them as the training data ($\mathbf{X}_i$ as its training features, and $\mathbf{y}_{ij}$ as its label). Eventually, well-trained logic learners for all concerned neurons based on

Algorithm 1 Logic Learner Training for Mirror Synthesizing

```
Input: input matrix $\mathbf{X}_i$ for each selected layer ($i = 1, 2, ..., k$) in $K$;
Input: weight $\mathbf{w}_{ij}$ and bias $b_{ij}$ for the $j^{th}$ neuron at the $i^{th}$ layer ($j = 1, 2, ..., t_i$);
Output: set of logic learner models $L$.
1: for $i = 1, 2, ..., k$ do          # traverse each FC layer
2:     for $j = 1, 2, ..., t_i$ do     # traverse each neuron of the layer
3:         $\mathbf{y}_{ij} := \mathbf{X}_i \mathbf{w}_{ij} - b_{ij} \cdot \mathbf{e}$
4:         $l_{ij} := meta\_model()$ # initialize a selected meta-model
5:         $l_{ij}.fit(\mathbf{X}_i, \mathbf{y}_{ij})$
6:         $L.append(l_{ij})$
7:     end for
8: end for
9: return $L$
```

logged information of training instances are returned (Line 9). By replacing concerned neurons in mirror M with new logic learners, VISION obtains its expected mirror model M.

C. Suitableness Measurement

Based on obtained mirror M, VISION in this phase evaluates its corresponding knowledge K's suitableness with respect to a given application scenario.

VISION uses behavioral differences between mirror M and knowledge K on a given application scenario to measure K's suitableness with respect to this scenario. This is based on the following *insight*: (1) if knowledge K extracted from the training scenario suits a new scenario, then both knowledge K and mirror M should behave similarly for the new scenario, since the training and new scenarios belong to the same type; (2) if knowledge K does not suit the new scenario, then knowledge K and mirror M should behave differently for the new scenario, since they agree only on the training scenario according to our model synthesis, not guaranteeing any performance for other different scenarios, and thus the likelihood that they behave coincidently similarly is low.

To measure such similarity, VISION models the behavior of a DNN model (knowledge K or mirror M) on predicting for a specific instance by intermediate outputs of concerned neurons in this model, i.e., a vector including all N concerned neurons' outputs in the form of $\mathbf{k} = \langle k_1, k_2, ..., k_N \rangle$. Then, based on this vector (named *behavior vector*), VISION measures the *behavioral differences* (BD score) between K and M by some distance function, e.g., traditional Euclidean distance. Let K's and M's behavior vectors on instance I be $\mathbf{k}_I$ and $\mathbf{m}_I$. Then the BD score can be calculated by formula $BD_I := \sqrt{(\mathbf{k}_I - \mathbf{m}_I)(\mathbf{k}_I - \mathbf{m}_I)^\top}$. Note that this treatment also addresses SynEva's flow problem, since SynEva measures the behavioral similarity via control-flow-alike paths of K and M, which can be tough to determine in DNN models.

Algorithm 2 calculates the BD score to evaluate the scenario suitableness for a given DNN model (knowledge K, accompanied with its corresponding mirror M). In practice, a BD score needs a threshold (*suitableness threshold*) for determining whether the score indicates "suitable" or "unsuitable", which is essentially application-specific. To address this trouble, VISION considers that a proper threshold should be around the upper bound of all possible BD scores for the training scenario, assuming that all instances from this scenario should be probably suitable. Therefore, we suggest to apply VISION to the training scenario first (i.e., calculating BD scores for knowledge K and mirror M), and VISION then uses this distribution of these calculated BD scores to decide the suitableness threshold value adaptively. Specifically, VISION refers to the definition of *upper bound* in drawing boxplots [24] to alleviate the disturbance of data outliers (named BD_{up}), and then decides the threshold: $\theta = p \times BD_{up}$ (p is a controllable parameter whose value is around one).

Algorithm 2 Suitableness Measurement of A Scenario

```
Input: one scenario $S = \{I_1, I_2, ..., I_n\}$ and its corresponding behavior vectors of $K$ and $M$: $\mathbf{k}_1, \mathbf{k}_2, ..., \mathbf{k}_n$; $\mathbf{m}_1, \mathbf{m}_2, ..., \mathbf{m}_n$;
Output: suitability analysis of instances in S.
1: for $i = 1, 2, ..., n$ do        # traverse each instance in $S$
2:     $BD_i := \sqrt{(\mathbf{k}_i - \mathbf{m}_i)(\mathbf{k}_i - \mathbf{m}_i)^\top}$ # Euclidean distance
3:     if $BD_i > \theta$ then
4:         $I_i$ is unsuitable
5:     else
6:         $I_i$ is suitable
7:     end if
8: end for
```

VISION then uses θ to distinguish suitable and unsuitable instances from application scenarios (Line 3-6). One can also evaluate suitableness for an application scenario by considering relative percentages between suitable and unsuitable instances from this scenario. A final question relates to the setting of the p value, which to some extent reflects the fitting degree between the selected logic learner meta-model in VISION and the training data from the training scenario. Generally, more overfitting the selected meta-model is, the larger p should be. Since VISION integrates several meta-models (owning different generalization abilities), their corresponding p values can vary. We suggest a range of $[0.6, 3.0]$ for the p value, which shows satisfactory effectiveness (accuracy of 76.4%–91.1%) in distinguishing suitable and unsuitable application scenarios in our later evaluation.

IV. EVALUATION

A. Research Questions

We answer the following three research questions:

- **RQ1:** *How effective is* VISION *in evaluating the suitableness of a trained DNN model against a given application scenario, as compared to SynEva?*
- **RQ2:** *How do different suitableness threshold values and logic learner meta-model choices affect* VISION*'s effectiveness?*
- **RQ3:** *What are* VISION*'s time and space overheads?*

B. Implementation

We implemented VISION in Python and its trained DNN models on Keras 2.2.4 [25] with Tensorflow 1.10.0 backend [26]. All DNN models were serialized and stored in the `.h5` file format for ease of usage. Based on the Keras interface, one can obtain intermediate information like weight, bias, and activation output values from certain neurons in a

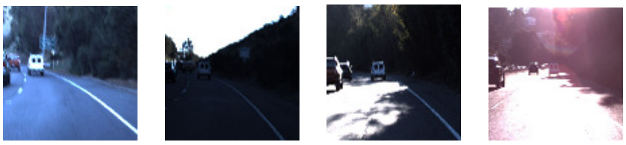
DNN model for use by Vision. For Vision's mirror model synthesis, its integrated logic learners were trained by Scikit-learn 0.19.1 [27], and the obtained models were also serialized and stored in the `.pkl` file format. For the efficiency concern, Vision trained logic learners in parallel.

For fair comparisons, we re-implemented SynEva also in Python and on Keras 2.2.4. Its integrated SVM logic learners were similarly serialized. To apply SynEva to DNN models, we conceptually mapped SynEva's control-flow-alike paths in the execution to a DNN model's activation paths in the prediction as suggested by DeepXplore [8].

C. Experimental Setup

Dataset. We selected a real-world self-driving dataset released by Udacity [20]. It contains images captured every 0.05 seconds by three cameras behind the windshield of a driving car, and simultaneous steering wheel angles applied by a human driver for each image. In total, the dataset has 107,010 samples, which are grouped by six sub-datasets. To facilitate experimental comparisons for different application scenarios, we partitioned these images captured by the central camera into four scenarios according to light conditions, namely, *clear*, *dark*, *shadow*, and *sunlight*, and resized them to `100*100`. Fig. 3 gives four examples, respectively. After removing really fuzzy samples (cannot be clearly classified even by humans), we obtained a total of 26,518 samples for experiments, as detailed in Table I. For these examples, we randomly divided them into a *training set* and a *test set* with a ratio of 9:1 for each scenario. Considering that the raw test sets were unbalanced for different scenarios, we rebalanced them for the same number of samples by random selection.

DNN models. We conducted experiments on DAVE-2, a widely-used end-to-end learning CNN architecture for predicting steering angles for self-driving cars from Nvidia [1], which includes five convolutional layers and five fully connected (FC) layers. We used the training sets from the four aforementioned scenarios to train and obtained four DNN models, namely, *clear-model*, *dark-model*, *shadow-model*, and *sunlight-model*.



(a) clear (b) dark (c) shadow (d) sunlight

Fig. 3. Four example scenarios.

TABLE I
Sizes of the datasets after partition

Scenario	Total size	Training set	Test set (unbalanced)	Test set (balanced)
clear	16,703	15,033	1,670	194
dark	4,353	3,918	435	194
shadow	3,514	3,163	351	194
sunlight	1,948	1,754	194	194
Total	26,518	23,868	2,650	776

TABLE II
Accuracy comparisons between SynEva and Vision

Approach	Trained DNN model	Actual label [*]	Predicted label [+]		Accuracy	Average accuracy
			US	*S*		
SynEva	clear-model	*US*	255	327	50.0%	69.8%
		S	61	133		
	dark-model	*US*	528	54	81.8%	
		S	87	107		
	shadow-model	*US*	460	122	71.0%	
		S	103	91		
	sunlight-model	*US*	580	2	76.5%	
		S	180	14		
Vision	clear-model	*US*	472	110	77.8%	82.3%
		S	62	132		
	dark-model	*US*	521	61	86.9%	
		S	41	153		
	shadow-model	*US*	492	90	75.6%	
		S	99	95		
	sunlight-model	*US*	535	47	89.0%	
		S	38	156		

[*] Oracle of instances. *S* refers to *suitable* (i.e., instances from the same scenario) and *US* refers to *unsuitable* (i.e., otherwise).

[+] Classification results predicted by SynEva and Vision.

D. Configuration

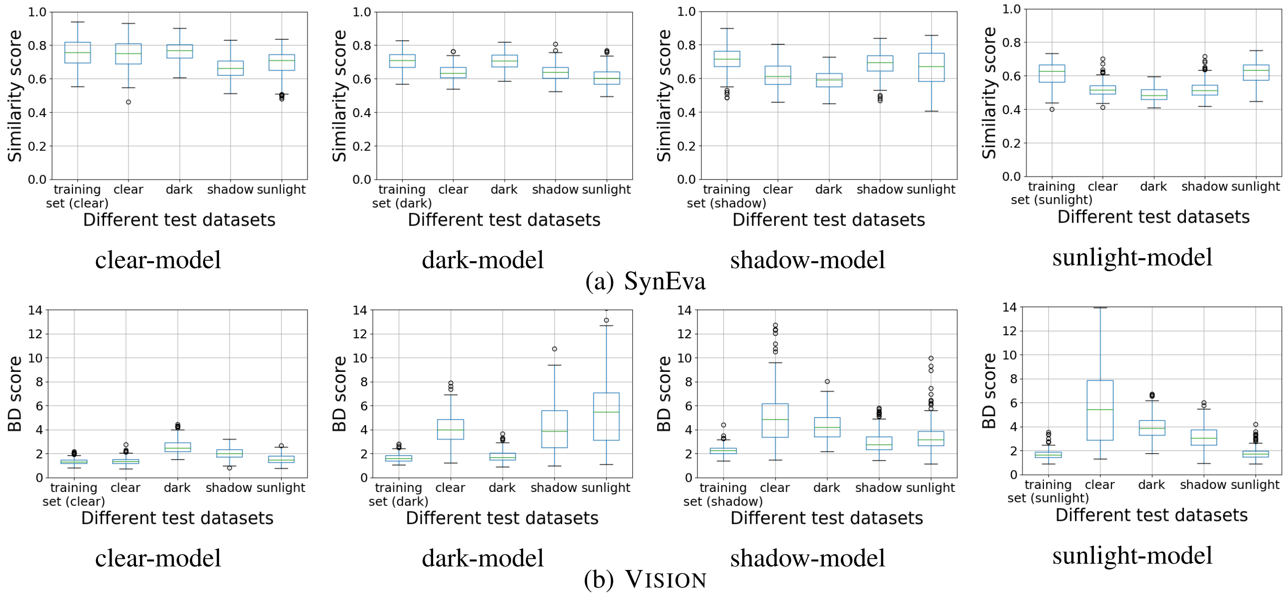
All experiments were conducted on a commodity PC with a four-core Intel(R) Core(TM) i7-6600 @2.60GHz CPU with 8GB RAM and a NVIDIA GeForce 930M GPU with 8GB RAM. The PC was installed with Microsoft Windows 10 Professional and Python 3.6.

E. Experimental Results and Analyses

In the following, we report and analyze experimental results, and answer the preceding three research questions in turn.

RQ1. We compare Vision and its predecessor SynEva in evaluating the suitableness of a DNN model against an application scenario. The comparisons were conducted on the four aforementioned trained DNN models, which were tested against each of the four testing scenarios (i.e., instances from the test sets in Table I), respectively, for comparing Vision's and SynEva's accuracies in the suitableness evaluation. The *accuracy* was calculated as the number of correct predictions (suitable or unsuitable) against all predictions. For comparing their potentials, we set the two approaches' parameters to their most proper values that can lead to the best effectiveness: (1) 0.80 for p in Vision and 0.70 for the similarity threshold in SynEva, and (2) SVR in Vision and SVM in SynEva (essentially SVR and SVM are the same algorithm). Note that since SynEva's original design did not work for all neurons (mirroring would take more than nine months roughly, and this suggests that our refinement for SynEva's overhead problem is indeed necessary), we made it choose neurons only from FC layers, similar to Vision, for the comparison purpose.

Table II lists the comparison results for each model (summing up all predictions from the four testing scenarios). We observe that: (1) SynEva's suitableness evaluation is unstable, with an accuracy ranging from 50.0% to 81.8% (mean: 69.8%); (2) Vision's suitableness evaluation is much more stable, ranging from 75.6% to 89.0% (mean: 82.3%); (3) in general, Vision outperformed SynEva by 12.5%, and the



clear-model dark-model shadow-model sunlight-model

(a) SynEva

clear-model dark-model shadow-model sunlight-model

(b) VISION

Fig. 4. Similarity and BD score distributions for the four DNN models, respectively (the training sets were sampled for the same data size as other test sets).

superiority was consistent (all positive) for all the four DNN models, namely, by 27.8%, 5.1%, 4.6%, and 12.5%.

To study VISION's superiority over SynEva, we looked into the distributions of their reported suitableness scores (i.e., BD scores in VISION and similarity scores in SynEva). Fig. 4 shows their score distributions. From the figure, we observe that VISION's BD scores clearly contributed to distinguishing suitable and unsuitable instances (i.e., disclosing that the instances in a test set was from the same or a different scenario, as compared to its corresponding training set), while for SynEva, its distinguishing ability was much weaker. This explains VISION's overall superiority in the suitableness evaluation over SynEva for all DNN models, and indicates that our treatment for SynEva's flow problem is effective.

Fig. 5 gives two illustrative examples reported by VISION, as one with the maximal BD score (most unsuitable) and one with the minimal BD score (most suitable) for the clear-model. One can easily observe that the second is under the clear weather condition, while the first is indeed far from clear.

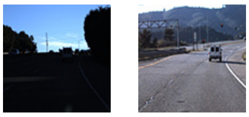
Therefore, we answer RQ1 as follows: VISION *was effective in evaluating a DNN model's suitableness for an application scenario, with an average accuracy of 82.3% (up to 89.0%), as compared to SynEva's 69.8%.*

RQ2. We next study the impact of different suitableness threshold values and logic learner meta-model choices on VISION's effectiveness. First, take the clear-model for example. We set VISION's logic learner meta-model to be DT and varied its p value in the $[0.6, 3.0]$ range, as mentioned earlier.

Fig. 6 illustrates how the p value impacted VISION's effectiveness (average accuracy). Roughly, the accuracy increased and then decreased, but within a small range. Although such fluctuation exists, the accuracy was mostly above 75% (more than 91% cases), which is generally fine. For other DNN models, the results were more or less the same.

Then we explore how different logic learner meta-model choices impact VISION's effectiveness. We focus on their respective best effectiveness within possible p values. Table III lists the comparison results among the six meta-models. We observe that: (1) VISION's best average accuracy varied from 76.4% to 91.1%; (2) among all the six meta-models, KNN achieved the highest best average accuracy of 91.1%, being 88.0–93.8% for the four DNN models, respectively; (3) non-linear meta-models (SVR, DT, and KNN) brought a higher best average accuracy, i.e., 82.3–91.1%, than linear meta-models (Ridge, LinearRegression, and LinearSVR), i.e., 76.4–79.5%. The last observation may be because inherent summation functions in DNN models are linear, and non-linear meta-models can generate learners with better generalization abilities, which can hardly behave coincidently similarly as the original DNN model when predicting new scenarios different from the training scenario.

Therefore, we answer RQ2 as follows: *Both suitableness threshold values and logic learner meta-model choices impacted* VISION*'s (best) effectiveness but not largely, and non-linear meta-models (especially KNN) brought higher accura-*



(a) Maximal (b) Minimal

Fig. 5. Examples with maximal/minimal BD scores against the clear-model

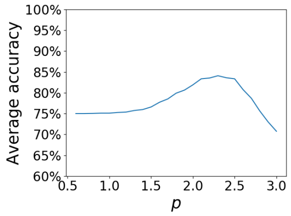


Fig. 6. Impact of different p values on VISION's average accuracy (with the DT meta-model and for the clear-model)

TABLE III
IMPACT OF DIFFERENT LOGIC LEARNER META-MODEL CHOICES

Meta-model	Trained DNN model	Actual label	Predicted label		Accuracy	Average accuracy
			US	*S*		
Ridge (p = 0.68)	clear-model	*US*	447	135	76.0%	78.6%
		S	51	143		
	dark-model	*US*	573	9	88.3%	
		S	82	112		
	shadow-model	*US*	571	11	75.3%	
		S	181	13		
	sunlight-model	*US*	582	0	75.0%	
		S	194	0		
LinearReg-ression (p = 0.78)	clear-model	*US*	452	130	77.7%	79.5%
		S	43	151		
	dark-model	*US*	570	12	88.9%	
		S	74	120		
	shadow-model	*US*	528	54	76.5%	
		S	128	66		
	sunlight-model	*US*	582	0	75.0%	
		S	194	0		
LinearSVR (p = 0.62)	clear-model	*US*	409	173	67.8%	76.4%
		S	77	117		
	dark-model	*US*	569	13	87.0%	
		S	88	106		
	shadow-model	*US*	546	36	75.6%	
		S	153	41		
	sunlight-model	*US*	582	0	75.0%	
		S	194	0		
SVR (p = 0.80)	clear-model	*US*	472	110	77.8%	82.3%
		S	62	132		
	dark-model	*US*	521	61	86.9%	
		S	41	153		
	shadow-model	*US*	492	90	75.6%	
		S	99	95		
	sunlight-model	*US*	535	47	89.0%	
		S	38	156		
DT (p = 2.30)	clear-model	*US*	557	25	87.2%	84.1%
		S	74	120		
	dark-model	*US*	533	49	83.5%	
		S	79	115		
	shadow-model	*US*	494	88	78.6%	
		S	78	116		
	sunlight-model	*US*	565	17	87.0%	
		S	84	110		
KNN (p = 0.85)	clear-model	*US*	572	10	92.8%	91.1%
		S	46	148		
	dark-model	*US*	540	42	90.0%	
		S	36	158		
	shadow-model	*US*	524	58	88.0%	
		S	35	159		
	sunlight-model	*US*	563	19	93.8%	
		S	29	165		

cies (82.3–91.1%) than linear ones (76.4–79.5%).

RQ3. Finally we study VISION's time and space overheads. We measured both the time for synthesizing mirrors (Phase 2 in Fig. 2) from DNN models, and that for suitableness evaluation (Phase 3 in Fig. 2). Besides, we measured the disk space consumed by synthesized mirrors with logic learners. Due to the space limit, we report results only for the dark-model (from 4,353 training instances) and sunlight testing scenario (with 194 instances) in Table IV (the second largest dataset). We observe that: (1) linear meta-models (Ridge, LinearRegression, and LinearSVR) consumed much less space (<16MB) than non-linear ones (SVR, DT, and KNN), where the KNN meta-model consumed the most space (94.4 GB); (2) these models' corresponding time overheads behaved similarly proportionally (except for SVR and KNN, whose synthesis time was almost the same); (3) although synthesis time overheads seemed large but they were for once only, while the evaluation time was much less (mostly $<$ 0.02 min per instance, except for KNN, which took 0.2 min per instance).

TABLE IV
VISION'S TIME AND SPACE OVERHEADS

Meta-model of Logic leaners	Synthesis time (min)	Evaluation time (min)	Consumed space
Ridge	4.13	0.04	8.3 MB
LinearRegression	48.45	0.05	15.8 MB
LinearSVR	35.82	0.07	16.0 MB
SVR	64.47	3.28	10.0 GB
DT	17.67	0.06	177.0 MB
KNN	63.83	42.75	94.4 GB

Combining both the effectiveness data in Table III and overhead data in Table IV, we would suggest DT as VISION's logic learner meta-model for practical usage, due to its balanced effectiveness (second highest best accuracy) and efficiency (similar evaluation time as linear meta-models, and acceptable synthesis time and consumed space). Futhermore, DT's superiority over SVR shows that our treatment for SynEva's choice problem is effective.

Therefore, we answer RQ3 as follows: *Both* VISION*'s time and space overheads varied with different logic learner meta-model choices (general acceptable), and the DT meta-model best balanced the effectiveness and efficiency (thus preferred).*

F. Threats Analysis

One possible threat concerns the internal validity of our experimental conclusions, i.e., different parameter values in training logic learners (e.g., k parameter for KNN and tree depth for DT) can lead to varying performance. To alleviate this threat, we mostly used default parameter values and restricted DT's maximal depth to be no more than 100. We consider such treatments reasonable. Besides, our focus is not to try all value combinations. Still, we obtained satisfactory results for suggesting VISION's practical usage.

V. RELATED WORK

Our work relates to quality assurance for DNN models. We discuss representative work in recent years on three aspects, namely, test adequacy, test input generation, and test oracle.

Test adequacy. Pei et al. [8] probably proposed the first test adequacy criterion, neuron coverage (NC) for testing DNN models. Ma et al. [9] extended NC with more fined-grained criteria to distinguish the major functional and corner behaviors of DNN models. Sun et al. [10] proposed four coverage criteria concerning distinct features of DNN models, inspired by the MC/DC. On the other hand, it was also argued that such structural coverage criteria for testing DNN models could be misleading if conducted when ignoring their usage contexts [11], and a better suggestion is to conduct the DNN testing under the operational context [28]. Kim et al. [16] further introduced a special surprise adequacy to measure the coverage of discretized input surprise ranges for DL systems based on kernel density estimation (KDE) and Euclidean distances. Similarly, our VISION measures the scenario suitableness also by distance, but focuses on the distance between knowledge and mirror programs from DNN models, not on the coverage of DNN models themselves.

Test input generation. DeepXplore [8] proposed a whitebox differential testing technique to generate inputs for testing DL systems. DeepTest [13] performed a greedy search with different image transformations for detecting erroneous behaviors on CNNs and Recurrent Neural Networks (RNNs). Zhang et al. [29] applied Generative Adversarial Network (GAN) to conduct driving-scenario oriented test generation with various artificial weather conditions. Instead of focusing on generating new or adversarial inputs, our VISION emphasizes evaluating existing inputs' suitableness from given application scenarios with respect to trained DNN models.

Test oracle. The oracle problem of DL programs is outstanding and challenging. One popular solution is to use metamorphic relations as alleviated test oracles, e.g., Ding et al. [30] proposed 11 metamorphic relations specially for testing DL systems. Differential testing is also another popular practice, which requests multiple implementations for the same or similar specification(s) for the comparison, e.g., DeepXplore [8] and DeepTest [13]. SynEva [17], the predecessor of VISION, instead uses behavioral differences between knowledge and mirror models as the pseudo-oracle for the comparison purpose. Our VISION inherits this nice idea, and further improves it with respect to DNN characteristics.

VI. CONCLUSION

In this paper, we address the scenario suitableness evaluation problem with respect to DNN models by proposing our VISION approach. VISION builds on SynEva, and explicitly addresses the latter's three limitations on the overhead, choice, and flow. Our experimental results show that VISION achieved a better accuracy of 4.6–27.8% than SynEva on the scenario suitableness evaluation regarding DNN models. Still, VISION has room for improvement. For example, its suitableness threshold should be more systematically obtained, and its logic learner meta-models should be further improved for reducing their training time and space overheads. We are working on them, as well as validating the VISION idea on more large-scale application scenarios.

ACKNOWLEDGMENT

This work was supported in part by National Key R&D Program (Grant #2017YFB1001801), National Natural Science Foundation (Grants #61932021, #61690204, #61802165) of China, and the Collaborative Innovation Center of Novel Software Technology and Industrialization, Jiangsu, China.

REFERENCES

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[2] I. Kononenko, "Machine learning for medical diagnosis: history, state of the art and perspective," *Artificial Intelligence in medicine*, vol. 23, no. 1, pp. 89–109, 2001.

[3] D. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *arXiv preprint arXiv:1202.2745*, 2012.

[4] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[5] S. S. Banerjee, S. Jha, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer, "Hands off the wheel in autonomous vehicles?: A systems perspective on over a million miles of field data," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 586–597.

[6] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[7] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[8] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated whitebox testing of deep learning systems," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 1–18.

[9] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu *et al.*, "DeepGauge: Multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 120–131.

[10] Y. Sun, X. Huang, and D. Kroening, "Testing deep neural networks," *arXiv preprint arXiv:1803.04792*, 2018.

[11] Z. Li, X. Ma, C. Xu, and C. Cao, "Structural coverage criteria for neural networks could be misleading," in *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results*. IEEE Press, 2019, pp. 89–92.

[12] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 3–29.

[13] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*. ACM, 2018, pp. 303–314.

[14] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "DLFuzz: Differential fuzzing testing of deep learning systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 739–743.

[15] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 39–57.

[16] J. Kim, R. Feldt, and S. Yoo, "Guiding deep learning system testing using surprise adequacy," in *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 2019, pp. 1039–1049.

[17] Y. Qin, H. Wang, C. Xu, X. Ma, and J. Lu, "SynEva: Evaluating ml programs by mirror program synthesis," in *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, July 2018, pp. 171–182.

[18] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[19] I. Smotroff, "Dataflow architectures: flexible platforms for neural network simulation," in *Advances in neural information processing systems*, 1990, pp. 818–825.

[20] Udacity, "self-driving car challenge," https://github.com/udacity/self-driving-car, 2016.

[21] F. R. Martins, E. B. Pereira, and R. A. Guarnieri, "Solar radiation forecast using artificial neural networks," *International Journal of Energy Science*, vol. 2, no. 6, 2012.

[22] P. Shrivastava, "Challenges in deep learning," https://hackernoon.com/challenges-in-deep-learning-57bbf6e73bb.

[23] "Fully connected layers in convolutional neural networks: The complete guide," https://missinglink.ai/guides/neural-network-concepts/fully-connected-layers-convolutional-neural-networks-complete-guide/.

[24] M. Frigge, D. C. Hoaglin, and B. Iglewicz, "Some implementations of the boxplot," *The American Statistician*, vol. 43, no. 1, pp. 50–54, 1989.

[25] F. Chollet *et al.*, "Keras," https://github.com/keras-team/keras, 2015.

[26] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[28] Z. Li, X. Ma, C. Xu, C. Cao, J. Xu, and J. Lü, "Boosting operational dnn testing efficiency through conditioning," *arXiv preprint arXiv:1906.02533*, 2019.

[29] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "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*. ACM, 2018, pp. 132–142.

[30] J. Ding, X. Kang, and X.-H. Hu, "Validating a deep learning framework by metamorphic testing," in *2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET)*. IEEE, 2017, pp. 28–34.