HKUST SPD - INSTITUTIONAL REPOSITORY

Title

Authors

Source

Version

DOI

Publisher

Copyright

This version is available at HKUST SPD - Institutional Repository (<https://repository.ust.hk/ir>)

If it is the author's pre-published version, changes introduced as a result of publishing processes such as copy-editing and formatting may not be reflected in this document. For a definitive version of this work, please refer to the published version.

EvalDNN: A Toolbox for Evaluating Deep Neural Network Models

Yongqiang Tian*
ytianas@cse.ust.hk
The Hong Kong University of Science
and Technology
Hong Kong, China

Zhihua Zeng*
zengzhihua@zju.edu.cn
Zhejiang University
Hangzhou, China

Ming Wen
mwena@hust.edu.cn
Huazhong University of Science and
Technology
Wuhan, China

Yepang Liu
liuy1@sustech.edu.cn
Southern University of Science and
Technology
Shenzhen, China

Tzu-yang Kuo
tkuo@connect.ust.hk
The Hong Kong University of Science
and Technology
Hong Kong, China

Shing-Chi, Cheung
scc@cse.ust.hk
The Hong Kong University of Science
and Technology
Hong Kong, China

ABSTRACT

Recent studies have shown that the performance of deep learning models should be evaluated using various important metrics such as robustness and neuron coverage, besides the widely-used prediction accuracy metric. However, major deep learning frameworks currently only provide APIs to evaluate a model's accuracy. In order to comprehensively assess a deep learning model, framework users and researchers often need to implement new metrics by themselves, which is a tedious job. What is worse, due to the large number of hyper-parameters and inadequate documentation, evaluation results of some deep learning models are hard to reproduce, especially when the models and metrics are both new.

To ease the model evaluation in deep learning systems, we have developed *EvalDNN*, a user-friendly and extensible toolbox supporting multiple frameworks and metrics with a set of carefully designed APIs. Using *EvalDNN*, evaluation of a pre-trained model with respect to different metrics can be done with a few lines of code. We have evaluated *EvalDNN* on 79 models from TensorFlow, Keras, GluonCV, and PyTorch. As a result of our effort made to reproduce the evaluation results of existing work, we release a performance benchmark of popular models, which can be a useful reference to facilitate future research. The tool and benchmark are available at <https://github.com/yqtianust/EvalDNN> and <https://yqtianust.github.io/EvalDNN-benchmark/>, respectively. A demo video of *EvalDNN* is available at: <https://youtu.be/v69bNjN2bJc>.

CCS CONCEPTS

• General and reference → Evaluation.

*Both authors contributed equally to this study.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

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

© 2020 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-7122-3/20/05.

<https://doi.org/10.1145/3377812.3382133>

KEYWORDS

Deep Learning Model, Evaluation

ACM Reference Format:

Yongqiang Tian, Zhihua Zeng, Ming Wen, Yepang Liu, Tzu-yang Kuo, and Shing-Chi, Cheung. 2020. *EvalDNN*: A Toolbox for Evaluating Deep Neural Network Models. In *42nd International Conference on Software Engineering Companion (ICSE '20 Companion)*, May 23–29, 2020, Seoul, Republic of Korea. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3377812.3382133>

1 INTRODUCTION

Recent studies [10, 12] showed that evaluating deep learning models' performance based on their accuracy alone is inadequate. A model achieving high accuracy could perform poorly in robustness [12] and fairness [6]. New metrics [7, 8, 10–12] have therefore been proposed for more comprehensive model evaluation. However, popular deep learning frameworks, such as TensorFlow [5] and PyTorch [4], only provide evaluation functions for accuracy. This causes the following problems.

First, users who want to deploy new metrics other than accuracy need to implement the published algorithms on top of the selected frameworks by themselves. The implementation is tedious because different frameworks are shipped with various APIs. Besides, the framework APIs have likely evolved significantly since the time when these algorithms were published. These induce the risk of API misuse, which can affect the model performance significantly [15]. Second, even if users find some existing implementations, for example from GitHub, they may not be able to use such implementations directly. For instance, although the authors of DeepXplore [12] provided their implementation of neuron coverage, their tool can only be used to evaluate Keras [1] models. If users want to assess a PyTorch model, they need to adapt the code of DeepXplore to PyTorch APIs accordingly. Third, models may not be reliably compared across different metrics implementations since it is hard to guarantee that different implementations would lead to the same evaluation results even for the same model.

In order to tackle the aforementioned problems, in this study, we design and implement *EvalDNN*, an open-source toolbox for Evaluating deep neural network (DNN) models. *EvalDNN* is designed to provide a unified programming interface to facilitate

the evaluation of DNN models from multiple perspectives across different frameworks. Specifically, it has the following features:

- (1) **Multiple frameworks support:** *EvalDNN* supports the evaluation of models built on top of four common frameworks: TensorFlow, PyTorch, MXNet [2], and Keras.
- (2) **Multiple metrics support:** *EvalDNN* supports three categories of evaluation metrics including accuracy, neuron coverage, and robustness.
- (3) **Fast coding:** *EvalDNN* provides simple and unified APIs for end users. Model evaluation can be implemented in just a few lines of code (See Section 3).
- (4) **Extensible to new metrics and frameworks:** *EvalDNN* is designed to be extensible and users can add new evaluation metrics and frameworks flexibly.

EvalDNN can benefit both developers and researchers in the deep learning communities. They can apply *EvalDNN* to evaluate the performance of models during development and deployment, using multiple metrics. With the provided unified APIs, they can also easily design and implement new evaluation metrics to extend *EvalDNN*. The unified APIs of *EvalDNN* can access a model's intermediate results to facilitate the computation of new metrics, without requiring users to learn different APIs provided by various frameworks (see Section 4.2.3). Moreover, with *EvalDNN*, evaluation for neuron coverage can be boosted up to five times (see Section 3).

The design of *EvalDNN* is described in Section 3. An example of implementing the evaluation of a deep learning model using a few lines of code with *EvalDNN* is also provided. To evaluate the correctness and reliability of our implementation, we selected 79 image classification models from four common model zoos. After comparing the results obtained by *EvalDNN* and those reported in literature, we found that 111 out of 158 results measured by *EvalDNN* in term of accuracy are the same as the originally reported ones. For the rest of them, the maximum difference is 1.1%. As for neuron coverage and robustness, the differences are marginal. Based on the evaluation results, we provide a reproducible benchmark (code available on our website), including the performance of 79 models in five metrics. These results are in Section 4.

In summary, our work makes two major contributions:

- (1) A extensible and user-friendly open-source toolbox for evaluating DNN models.
- (2) A reproducible benchmark consisting of the evaluation results of 79 pre-trained models in terms of multiple metrics.

2 RELATED WORK

There are some projects targeting on evaluating DNN models. For example, DeepSec [9] is a unified platform for evaluating the robustness of PyTorch models. A recent study conducted a survey to explore the relation between accuracy and robustness [14], and their proposed tool can be leveraged to evaluate the robustness of TensorFlow models. Besides, Foolbox [13] is designed to generate adversarial examples for DNN models and thus can be used to evaluate the models' robustness. However, these projects only target at the evaluation of deep learning systems with respect to a specific metric or framework, while *EvalDNN* supports various metrics cross different frameworks. In this study, we also leverage Foolbox to generate adversarial examples when implementing *EvalDNN*.

Figure 1: Overview of *EvalDNN*

3 DESIGN OF THE TOOL

The architecture of *EvalDNN* is shown in Figure 1. *EvalDNN* encompasses a set of unified interfaces to evaluate a given model. Such interfaces can also access a model's output and intermediate status to extract necessary information for evaluation. Users can easily extend *EvalDNN* based on the given interfaces to support other deep learning frameworks, or new evaluation metrics (see Section 4.2.3), in addition to those listed in Figure 1. The unified interfaces also enhance the independency between framework and metric. For instance, when users add a new metric for evaluation, they do not need to consider the API difference across over frameworks.

Listing 1 shows a usage example of *EvalDNN*. To use *EvalDNN*, users only need to feed the target DNN model and dataset into *EvalDNN* (lines 3–4), and specify the target evaluation metric (lines 6–7). *EvalDNN* will then evaluate the given model and generate the corresponding report (line 9). As shown in Figure 1, *EvalDNN* provides users with unified interfaces, which support different types of frameworks, various evaluation metrics and datasets. Specifically, it supports four deep learning frameworks: TensorFlow, Pytorch, MXNet, and Keras. As for metrics, *EvalDNN* currently supports five metrics from three categories, as listed in Table 1.

```

1 # import tool is ignored
2 # load model and data
3 model = keras.applications.VGG16()
4 data = load_data()
5 # wrap model and specify the metric
6 measure_model = KerasModel(model)
7 neuron_coverage = NeuronCoverage()
8 #evaluation
9 measure_model.intermediate_layer_outputs(data.x, [
    neuron_coverage.update, neuron_coverage.report])

```

Listing 1: An Example to Evaluate VGG16 with Respect to Neuron Coverage using *EvalDNN*

One of the biggest engineering challenges in the implementation is that the original implementation of neuron coverage [12] is inefficient. Specifically, it takes around 3.5 hours to evaluate Keras vgg16 model on the ImageNet validation set. To tackle this problem, we carefully re-design the computation flow and leverage a high performance python compiler [3] to parallelize the computation on multiple native threads. As a result, the evaluation process of neuron coverage is accelerated by five times. The evaluation time aforementioned is reduced to only 40 mins. Hopefully, our approach would make neuron coverage more practical for model evaluation.

4 BENCHMARK AND EVALUATION

Based on the unified interfaces provided by *EvalDNN*, we evaluated 79 models from the common model zoos. The purpose is to

Table 1: Embedded Metrics in EvalDNN

Categories	Metrics	Descriptions
Accuracy	Top-K Accuracy	The fraction of inputs for which the ground truth is among the top-k probable labels in the model output.
Neuron Coverage	Neuron Coverage	The fraction of neurons that are activated during the model inference stages.
Robustness	Success Rate	The percentage of adversarial attacks that are successful.
	Time	The duration to generate an adversarial sample.
	Distance	The distance between the original input and the corresponding adversarial sample.

evaluate the correctness and reliability of the implementation of *EvalDNN*. Besides, the results of our experiments can also provide a benchmark, including the evaluation results of the 79 models with respect to various metrics, for future studies.

4.1 Experiment Setup

Model Selection. We first selected four popular model zoos, namely, TensorFlow-Slim,¹ TorchVision,² GluonCV,³ and Keras.⁴ We then selected 79 models targeted for image classification from these model zoos, following the three steps below. We only focus on image classification tasks in the experiments because most models in the above model zoos are designed for image classification.

- (1) We first exclude those models, of which the implementation is unavailable. For example, ResNetV2-200 in TensorFlow-Slim is excluded since its model file is not available.
- (2) In each model zoo, if a type of models has no more than three variants, we select all of them for our experiments. For example, in Keras, model type of VGG has only two variants, i.e., VGG-16 and VGG-19, and we selected both.
- (3) If a type of models has more than three variants, we only select three of them. Specifically, we give higher priorities to those variants that are also available in the other model zoos. For example, in GluonCV, DenseNet has 4 variants, i.e., DenseNet-121/161/169/201. We only selected DenseNet-121/169/201 since they are also available in all the other three model zoos while DenseNet-161 is only available in Torch.

Evaluation: Our experiments are conducted on Azure NC6 instances with K80 GPU, using Python 3.7.5, TensorFlow 1.15, PyTorch 1.3.1, MXNet 1.5.1, GluonCV 0.5.0, Keras 2.3.1.

The experiments are conducted using ImageNet Val 2012 dataset. We compute various evaluation metrics as follows. For neuron coverage, we select the activation threshold from 0.0 to 0.9 with step 0.1. Certain layers have been filtered out, following the original implementation of DeepXplore, such as the input layer and flatten layer in Keras model. To evaluate robustness, we select three adversarial attack methods: FGSM [7], BIM [8], and DeepFool [11]. Then we randomly choose 1,000 images from the whole set and use the above three methods to perform attacks on these images. The IDs of selected images are published on our website. For each model, the misclassified images are not involved in adversarial attack.

4.2 Results and Discussions

We only show partial results of our experiments in Table 2 due to the page limit. The full results are available on our website. For accuracy, the declared accuracy in model zoos is shown in parentheses. For neuron coverage, we also show the results provided by the original paper [10] in parentheses if they are available. Note that they used a subset of dataset while we use the full set. For robustness, we leveraged the tool published by a recent survey [14] and conducted the similar experiments for cross validation. The results are shown in parentheses. Although their tool can only support part of the TensorFlow Slim models in our benchmark, we didn't find a more comprehensive robustness benchmark for ImageNet.

4.2.1 Reliability of implementation. As shown in Table 2, 24 of the 32 accuracy values obtained by *EvalDNN* are the same as the ones declared in model zoos. Six of the 32 values are relatively lower than the reported ones, but the differences are within 0.8%. The remaining two accuracy values measured by *EvalDNN* are higher than the reported ones. In terms of all 79 models, the evaluation results obtained by *EvalDNN* with respect to accuracy are the same as the reported ones for 111 out of 158 cases. The maximum difference is 1.1% and most of the differences are within 0.5%. The difference could be caused by experiment environment, such as software version. For neuron coverage, the results obtained by *EvalDNN* is relative higher compared with the ones reported by a recent study [10]. The major reason is that the existing study [10] only used 5,000 out of the 50,000 images while we use all of them in our experiments. As for robustness, the difference between the results from *EvalDNN* and baseline are very marginal. For all the 14 models that are available both in our benchmark and [14], the maximum difference of success rate is up to 0.3%. Such differences might be caused by different random seeds adopted in sample selection. The results demonstrate that our implementation is reliable.

4.2.2 Fast coding. Listing 1 shows that users can finish the evaluation of deep learning models within a few lines of code. Specifically, after loading model and data, it takes two lines to wrap the model and specify the evaluation metric (i.e., accuracy) and another single line to perform the evaluation. For certain models built on TensorFlow, one more line is needed to close the computation session. In general, for any pre-trained model built on the four frameworks, with respect to any single metric, the implementation could be finished within less than five lines of codes, excluding the data preprocessing and model loading. In contrast, if developers plan to evaluate a model with respect to neuron coverage without *EvalDNN*, they need to adapt the implementation provided by DeepXplore. Specifically, it requires developers to understand the source code

¹<https://github.com/tensorflow/models/tree/master/research/slim>

²<https://pytorch.org/docs/stable/torchvision/models.html>

³https://gluon-cv.mxnet.io/model_zoo/classification.html

⁴<https://keras.io/applications/>

Table 2: Benchmark: Evaluation Results of Selected Models in Model Zoos

Model Zoo	Model	Accuracy		Neuron Coverage Threshold 0.5	Robustness (FGSM)		
		Top-1	Top-5		Success Rate	Average Time	Average L_∞ Distance
TensorFlow Slim	vgg16	70.9%(71.5%)	89.8% (89.8%)	80.0%	100%(99.9%)	0.81s	0.0042 (0.0044)
	vgg19	71.0%(71.1%)	89.8%(89.8%)	77.2%	100% (99.9%)	1.01s	0.0052 (0.0044)
	resnet_v1_50	75.2%(75.2%)	92.2%(92.2%)	71.9%	100%	0.42s	0.0077
	mobilenet_v2_1_0_224	71.8%(71.9%)	90.7%(91.0%)	78.2%	100%	0.16s	0.0083
TorchVision	vgg16	71.6%(71.6%)	90.4%(91.3%)	64.1%	100%	0.77s	0.0061
	vgg19	72.4%(72.4%)	90.9%(90.9%)	59.0%	100%	1.04s	0.0094
	resnet50	76.1%(76.1%)	92.9%(92.9%)	48.4%	99.9%	0.57s	0.0105
	mobilenet_v2	71.9%(71.9%)	90.3%(90.3%)	51.3%	100%	0.24s	0.0025
GluonCV	vgg16	73.2%(73.2%)	91.3%(91.3%)	79.9%	100%	0.63s	0.0307
	vgg19	74.1%(74.1%)	91.8%(91.4%)	77.5%	100%	0.88s	0.0111
	resnet50	77.4%(77.4%)	93.6% (93.6%)	89.0%	100%	1.87s	0.0828
	mobilenet_v2_1_0	72.0%(72.0%)	90.6% (90.6%)	100.0%	99.9%	1.71s	0.0906
Keras	vgg16	71.3%(71.3%)	90.0%(90.1%)	63.4%	100%	0.71s	0.0052
	vgg19	71.3%(71.3%)	90.0%(90.0%)	58.5%(57.2%)	100%	0.90s	0.0060
	resnet50_v1	74.9%(74.9%)	92.1%(92.1%)	75.7%(70.6%)	100%	0.58s	0.0073
	mobilenet_v2	71.2%(71.3%)	90.3%(90.1%)	76.9%	100%	0.29s	0.0073

Reference value is in parentheses if available. The cell is **bold** if result from *EvalDNN* is different with reference value.

and identify the necessary APIs to compute the neuron coverage, which might involves a few hours and tens of code. The evaluation on robustness requires more labor work. This demonstrates that *EvalDNN* is a fast-coding model evaluation tool for users.

4.2.3 Extensibility. *EvalDNN* is designed to be extensible for supporting new metrics and frameworks. For instance, suppose users want to implement a new metric called *frequently-activated neuron coverage*, to measure the percentage of neurons that are activated frequently. Using *EvalDNN*, they only need to implement a new class under *eval.metrics*, of which only a callback function has to be defined. The callback function can then take the outputs from model's intermediate layers as input and calculate the new metric accordingly. The implementation is independent of different deep learning frameworks and thus users do not need to handle the various APIs and data structures provided by these frameworks. To use the new evaluation metric, the users only need to pass the callback function to *EvalDNN*, similar to line 9 in Listing 1. *EvalDNN* will then pass the outputs of intermediate layers to the callback functions for computation. Once the metric is implemented, it can be used for any model provided that its framework is supported by *EvalDNN*. Such elegant design of *EvalDNN* enables its extensibility.

5 ACKNOWLEDGMENT

Part of this work was conducted during Zhihua Zeng's internship at HKUST. This project is sponsored by the National Natural Science Foundation of China (No. 61932021 and 61802164), Microsoft Cloud Research Software Fellow Award 2019, Hong Kong PhD Fellowship Scheme, and MSRA Collaborative Research Grant.

6 CONCLUSION

In this study, we present *EvalDNN*, a user-friendly and extensible toolbox to facilitate the evaluation of DNN models. A reproducible benchmark containing the evaluation results of 79 models using multiple metrics is also provided for further study. In the evaluation, we showed that *EvalDNN* is reliable, fast coding and extensible.

REFERENCES

- [1] [n.d.]. Keras. <https://keras.io/>. Accessed: 2019-12-06.
- [2] [n.d.]. MXNet. <https://mxnet.apache.org/>. Accessed: 2019-12-06.
- [3] [n.d.]. Numba. <http://numba.pydata.org/>. Accessed: 2019-12-06.
- [4] [n.d.]. PyTorch. <https://pytorch.org/>. Accessed: 2019-12-06.
- [5] [n.d.]. TensorFlow. <https://www.tensorflow.org/>. Accessed: 2019-12-06.
- [6] Aniya Aggarwal, Pranay Lohia, Seema Nagar, Kuntal Dey, and Diptikalyan Saha. 2019. Black Box Fairness Testing of Machine Learning Models. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019)*. ACM, New York, NY, USA, 625–635. <https://doi.org/10.1145/3338906.3338937>
- [7] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). <http://arxiv.org/abs/1412.6572>
- [8] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. [arXiv:cs.CV/1607.02533](https://arxiv.org/abs/1607.02533)
- [9] X. Ling, S. Ji, J. Zou, J. Wang, C. Wu, B. Li, and T. Wang. 2019. DEEPSEC: A Uniform Platform for Security Analysis of Deep Learning Model. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 673–690. <https://doi.org/10.1109/SP.2019.00023>
- [10] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018. DeepGauge: Multi-granularity Testing Criteria for Deep Learning Systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*. ACM, New York, NY, USA, 120–131. <https://doi.org/10.1145/3238147.3238202>
- [11] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2574–2582. <https://doi.org/10.1109/CVPR.2016.282>
- [12] Kexin Pei, Yinshi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM, New York, NY, USA, 1–18. <https://doi.org/10.1145/3132747.3132785>
- [13] Jonas Rauber, Wieland Brendel, and Matthias Bethge. 2017. Foolbox v0.8.0: A Python toolbox to benchmark the robustness of machine learning models. [CoRR abs/1707.04131](https://arxiv.org/abs/1707.04131) (2017). [arXiv:1707.04131](https://arxiv.org/abs/1707.04131) <http://arxiv.org/abs/1707.04131>
- [14] Dong Su, Huan Zhang, Hongge Chen, Jinfeng Yi, Pin-Yu Chen, and Yupeng Gao. 2018. Is Robustness the Cost of Accuracy? – A Comprehensive Study on the Robustness of 18 Deep Image Classification Models. In *The European Conference on Computer Vision (ECCV)*.
- [15] Yuhao Zhang, Yifan Chen, Shing-Chi Cheung, Yingfei Xiong, and Lu Zhang. 2018. An Empirical Study on TensorFlow Program Bugs. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2018)*. ACM, New York, NY, USA, 129–140. <https://doi.org/10.1145/3213846.3213866>