PyTorch-OOD: A Library for Out-of-Distribution Detection based on PyTorch

Konstantin Kirchheim Marco Filax Frank Ortmeier

Department of Computer Science
Otto-von-Guericke University Magdeburg
{firstname.lastname}@ovgu.de

Abstract

Machine Learning models based on Deep Neural Networks behave unpredictably when presented with inputs that do not stem from the training distribution and sometimes make egregiously wrong predictions with high confidence. This property undermines the trustworthiness of systems depending on such models and potentially threatens the safety of their users. Out-of-Distribution (OOD) detection mechanisms can be used to prevent errors by detecting inputs that are so dissimilar from the training set that the model can not be expected to make reliable predictions. In this paper, we present PyTorch-OOD, a Python library for OOD detection based on PyTorch. Its primary goals are to accelerate OOD detection research and improve the reproducibility and comparability of experiments. PyTorch-OOD provides well-tested and documented implementations of OOD detection methods with a unified interface, as well as training and benchmark datasets, architectures, pre-trained models, and utility functions. The library is available online[1] under the permissive Apache 2.0 license and can be installed via Python Package Index (PyPI).

1. Introduction

Due to their versatility and performance, Machine Learning models based on Deep Neural Networks (DNN) [29, 43] are increasingly deployed in products and intelligent services across a wide range of fields, including healthcare [13], automatic content filtering [49], collaborative robotics [26], and financial services [9]. However, despite their remarkable performance in controlled environments, DNN-based models are demonstrably not robust to shifts in the data generating distribution, and as a result, their predictions can not always be trusted. In safety-critical applications, undetected errors can pose a threat to the psychological and physical well-being of humans and, consequentially, have to be avoided. Out-of-Distribution (OOD) detection, which is concerned with detecting inputs that have low probability under the training distribution, has gained remarkable attention in recent years. There are several closely related fields, like Open Set Recognition (OSR) [16], Uncertainty Estimation [15], Novelty Detection [38], Anomaly Detection [39], and Outlier Detection [1], and the terms are sometimes used interchangeably.

In this work, we present PyTorch-OOD, a library for Out-of-Distribution detection in high dimensional data based on the PyTorch deep learning framework [37]. PyTorch-OOD aims to provide modular, well-tested, and documented implementations of OOD detection methods with a unified interface, as well as training and benchmark datasets, pre-trained models, and utility functions. Since the boundaries to adjacent fields, like OSR, are sometimes blurred, the library also covers methods from closely related fields.

The remainder of this paper is structured as follows: The goals of PyTorch-OOD are described in Section 2. The design decision and fundamental assumptions are presented in Section 3. An overview of the library's content is given in Section 4. A study of several state-of-the-art OOD detection methods and datasets is presented in Section 5. In Section 6, we briefly survey related work and close with a conclusion and an outlook on future work in Section 7.

2. Goals

PyTorch-OOD is designed with the following goals:

Promoting Reproducibility Recent works in several machine learning domains have noted difficulties in reproducing experiments, including supervised classification [4, 45], reinforcement learning [35], and unsupervised OOD detection [27]. These difficulties are attributed, among others, to intrinsic and extrinsic sources of nondeterminism. Intrinsic sources include all factors that can lead to different results when using otherwise identical code, for example, parameter initialization, dropout, the shuffling of training data, or

[1] https://gitlab.com/kkirchheim/pytorch-ood

the sampling of datasets. While intrinsic nondeterminism can theoretically be controlled by setting a fixed random seed (or disabling nondeterministic operations in the case of low-level libraries like cuDNN [6]), such an approach may also reduce the validity of conclusions to this random seed [4], as minor differences in the initial conditions can cause significant differences in experimental outcomes [45]. By providing a library with well-tested and documented implementations of existing methods, we aim to mitigate the effects of extrinsic sources of nondeterminism. Apart from detection methods, there are many different benchmark tasks used in the academic literature that currently lack access through a unified interface. By providing easily accessible and unified implementations, we hope to encourage researchers to subject newly proposed methods to a broader range of benchmarks tasks and comparison with existing methods, which will provide a more comprehensive picture of their performance.

Accelerating Research PyTorch-OOD aims to increase the speed with which research hypotheses can be tested by facilitating fast prototyping. This requires minimizing manual intervention for tasks that can be automated, maintaining consistent interfaces, and ensuring compatibility with other software frameworks. Furthermore, PyTorch-OOD aims to eliminate boilerplate code without making too strong assumptions on OOD detection experiments.

3. Design

PyTorch-OOD makes the following assumptions:

Binary Classification PyTorch-OOD casts OOD detection as binary classification with the goal to discriminate between in-distribution (IN) and out-of-distribution (OOD) data. This detection is performed in addition to other tasks, like classification or segmentation. Thus, it is assumed that each OOD detector produces outlier scores, where high values indicate greater outlierness. While these are strong assumptions that some detectors, like OpenMax [2], might not adhere to, we argue that most methods can be reformulated in such a fashion.

Workflow PyTorch-OOD adopts a workflow that is motivated by the empirical observation that many scientific publications follow a three-staged approach when training and evaluating OOD detection methods:

1. A DNN is trained on some dataset(s) for some task(s),
2. an OOD detector is constructed based on the DNN. The detector might require fitting to some dataset(s), and finally
3. the detector is evaluated on some dataset(s).

This workflow is illustrated in Figure 1. In PyTorch-OOD, all of these steps are decoupled. PyTorch-OOD makes no assumptions about the DNN architecture or the used training procedure.

Labeling PyTorch-OOD assumes that all IN samples have positive class labels ≥ 0 and all OOD samples negative ones. This allows to efficiently discriminate such samples dynamically at runtime, making it possible to work with datasets containing both IN and OOD data. Furthermore, this assumption allows to design DNN modules that automatically handle OOD data differently between training and testing, without the need for complex branching or redundancy that would be required otherwise.

Interface PyTorch-OOD uses PyTorch abstractions where possible and a Scikit-Learn-like interface everywhere else [5]. In particular, PyTorch-OOD does not imitate more extensive frameworks that facilitate scalability, but aims to integrate with them seamlessly. Generally, an OOD detector has two methods:

- `detector.fit(data)`
- `detector.predict(batch)`

The optional `fit` method is used to fit the detector to a specific dataset, for example, to determine the parameters of an underlying statistical model. Examples of such methods that require fitting after the initial network optimization include the Mahalanobis OOD Detector [30] and the OpenMax Layer [2]. The `predict` method takes a batch of data and returns a tensor with outlier scores. This interface design allows for interoperability with the PyTorch software ecosystem, for example, PyTorch Lightning[2] and TorchMetrics[3].

4. Library Content

This section provides an overview of the libraries content at the time of writing.

4.1. Objective Functions

NNs are often used to learn a hierarchy of increasingly lower dimensional representations of the data. Their training usually involves searching for a set of parameters that empirically minimize the expected value of some objective function. PyTorch-OOD provides implementations of several objective functions that have been proposed to learn representations that allow to discriminate between IN and OOD data more effectively. Objective functions are loosely categorized into unsupervised and supervised. Examples are listed in Table 1.

[2] https://www.pytorchlightning.ai
[3] https://torchmetrics.readthedocs.io

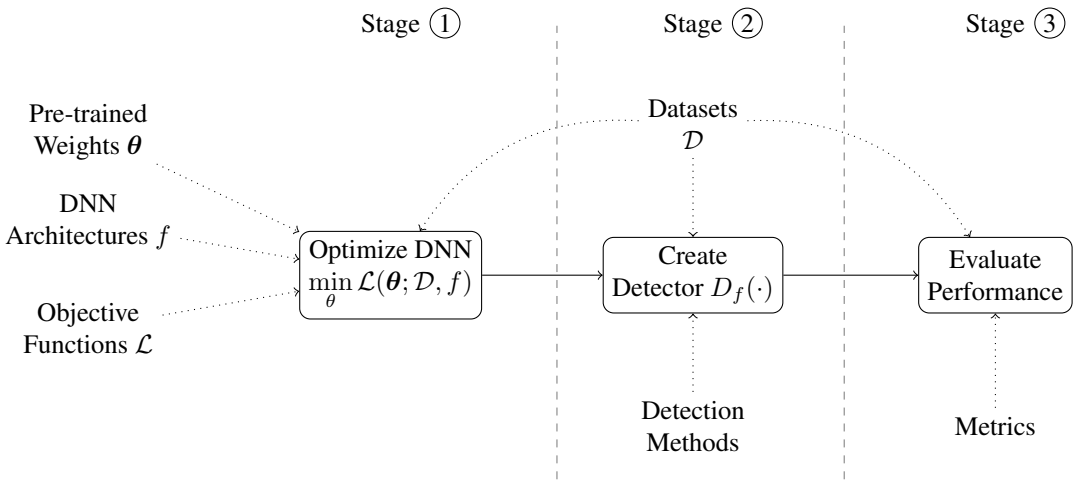

Figure 1. Depiction of PyTorch-OODs workflow, showing how components of the library can be used throughout the different stages: (1) A DNN is optimized in order to learn a lower dimensional representation of the input data. (2) A OOD detection model is constructed, based on the DNN and optionally a dataset. (3) The OOD detection model is tested on a dataset to discriminate between IN and OOD samples.

Unsupervised Unsupervised objective functions only use IN data during training. In our implementation, they automatically ignore OOD inputs. This allows composing more complex objective functions from supervised and unsupervised objective functions without the need for additional modification.

Supervised Supervised objective functions use both IN and OOD data during training, usually with the goal to preserve the discriminability of IN and OOD data in the lower dimensional output space of the DNN. Often, they constitute supervised generalizations of unsupervised objective functions such that they can be implemented as a linear combination of several objective function modules.

4.2. Detection Methods

After the DNN has been trained, an OOD detection model is created. This can be seen as constructing a model of normality, for example, a density model of normal representations, a model of distances from some class prototypes, or a model of reconstruction errors. For some instance $\mathbf{x}$, these methods then yield an outlier score $D_f(\mathbf{x})$. A threshold τ can be applied to this score in order to discriminate samples into OOD and IN:

$$\text{outlier}(\mathbf{x}) = \begin{cases} 1 & \text{if} \quad D_f(\mathbf{x}) > \tau \\ 0 & \text{else} \end{cases} . \tag{1}$$

A selection of the implemented detectors is listed in Table 2.

4.3. Datasets

Data is an essential ingredient for Machine Learning, but downloading and preparing datasets can be tedious. While

Table 1. Selection of implemented objective functions.

Objective Function	**Note**
Unsupervised	
Cross-Entropy	
II Loss [18]	Class Prototype-based
Center Loss [47]	Class Prototype-based
CAC Loss [33]	Class Prototype-based
Supervised	
Deep SS-SVDD [41]	One-Class Method
Energy Regularized [32]	Maximizes Energy Score gap between IN and OOD
Entropic Open-Set [11]	Minimizes $\|f(\mathbf{x})\|_2^2$ for OOD
Objectosphere [11]	Increases difference of $\|f(\mathbf{x})\|_2^2$ for IN and OOD
Outlier Exposure [24]	Reduces Softmax Score for OOD data

PyTorch provides code that automates these tasks for several datasets, at the moment, several datasets that are commonly used in OOD detection benchmarking have to be manually set up. PyTorch-OOD does not introduce new benchmark datasets, but instead provides easy access to different datasets that are frequently used throughout the literature, by eliminating the need to manually download the data and writing code for loading it. Currently, PyTorch-OOD includes datasets for Computer Vision and Natural Language Processing that can be used for pre-training, as well as for benchmarking OOD detectors. Access to pre-training can support research

Table 2. Selection of implemented OOD detection methods.

OOD Detector	Note
Softmax [22]	Baseline Method
OpenMax [2]	Distance-based
Mahalanobis [30]	Distance-based
Odin [31]	Preprocessing Method
Deep SVDD [40]	Distance-based One-Class Method
Energy-based [32]	
Monte Carlo Dropout [15]	Probabilistic

on supervised objective functions. Furthermore, since OOD detection methods are usually tested agains many different datasets, having access to a larger collection of benchmarking datasets allows researchers to evaluation their methods on a broader range of benchmark tasks in order to provide a more comprehensive picture of the performance of newly proposed methods. An overview of the datasets is provided in Table 3. Datasets can be roughly divided into *natural*, *corrupted* and *synthetic*.

Natural Achieving state-of-the-art performance often requires pre-training or supervision on large quantities of real-world data, like the ImageNet [10]. A dataset commonly used for supervised methods, the TinyImages database [46], has recently been taken down due to ethical concerns [3]. Consequentially, several OOD detection methods can not be reproduced [48]. PyTorch-OOD provides code for the automatic set up of alternative datasets, including a cleaned version of the original TinyImages database provided by [24] that consists of 300.000 images.

Corrupted Several datasets include corrupted versions of natural data, for example, images overlaid by noise or more complex corruptions, that can be used to benchmark model robustness [21]. While corruptions could also be applied dynamically at runtime, using static datasets is advisable considering reproducibility.

Synthetic Several datasets include fully synthetic data. These can be sampled from some generative model, for example, different kinds of noise, or created by other means like evolutionary algorithms [36].

4.4. Deep Neural Networks

The objective function and the detection method often constitute the central innovation of publications in the OOD literature. Often, the conducted experiments build on existing DNN architectures, and sometimes, methods also reuse weights from other publications. However, at the moment,

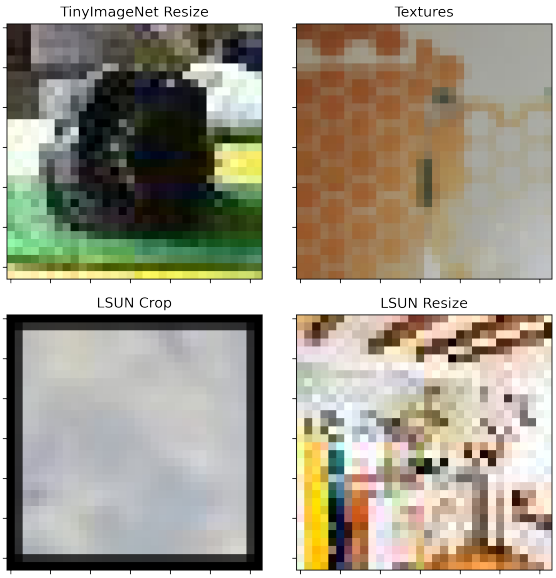

Figure 2. Examples of Out-of-Distribution Images from different benchmark datasets for CIFAR models.

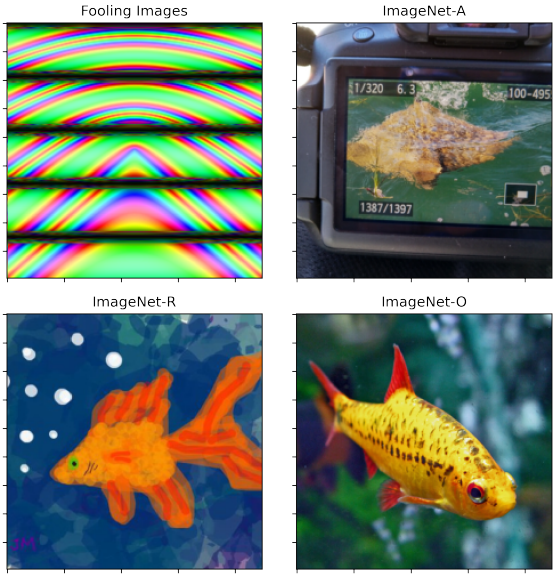

Figure 3. Examples of Out-of-Distribution Images from different benchmark datasets for ImageNet models.

implementations of architectures and pre-trained weights used for OOD detection are scattered across different codebases.

Architectures While PyTorch provides standardized implementations of several models, OOD experiments often involve custom implementations of DNNs, making it difficult to reproduce exact numerical results without access

Table 3. Selection of supported datasets. The DL column indicates automatic download capability.

Dataset	Note	DL
Pre-Training		
TinyImages 300k [46]	A cleaned version of the 80 Million Tiny Images Dataset with 300.000 images.	✓
ImageNet-DS [7]	A downsampled version of the ImageNet.	✗
Benchmark		
MNIST-C [34]	Corrupted MNIST	✓
ImageNet-A [25]	Natural Adversarial Examples	✓
ImageNet-O [25]	OOD for ImageNet	✓
ImageNet-R [20]	Rendition Images	✓
ImageNet-C [21]	Corrupted ImageNet	✓
CIFAR 10-C [21]	Corrupted CIFAR 10	✓
CIFAR 100-C [21]	Corrupted CIFAR 100	✓
Fooling Images [36]	Synthetic Images	✓
Textures [8]	Images used in [30–32]	✓
TinyImageNet Crop	Images used in [30–32]	✓
TinyImageNet Resize	Images used in [30–32]	✓
LSUN Crop	Images used in [30–32]	✓
LSUN Resize	Images used in [30–32]	✓
Noise	Gaussian, Uniform, ...	✓
Newsgroup-20	Texts used in [21]	✓
Reuters-52	Texts used in [21]	✓
Multi30k	Texts used in [21]	✓
WMT16	Texts used in [21]	✓
WikiText-2	Texts used in [24]	✓

to the original code. For example, several publications use custom implementation of a WideResNet [50] with 40 Layers [22, 23, 30, 32]. PyTorch-OOD, therefore, provides code for DNN architectures that are frequently used but not included in other packages to facilitate comparability and reproducibility of results.

Pre-Trained Weights Several publications provide weights for DNNs that are pre-trained on large datasets of IN or OOD data. Having access to such models serves several purposes. Firstly, pre-trained models can be used to easily compare newly proposed methods or to reevaluate methods on new datasets. Secondly, pre-trained feature encoders can be used for subsequent finetuning on downstream tasks. It has been shown that pre-training on large datasets can significantly increase the robustness of models [14, 23], and [23] argues that pre-training should be used consistently in OOD detection.

Table 4. Selection of architectures. The PT column indicates the availability of pre-trained weights.

Model	Note	PT
WideResNet	A Wide Residual Network used by [22, 23, 30, 32]	✓
ViT	Vision Transformer used by OOD-Former [28]	✓
Base GRU	GRU-based model as used in [24]	✗

Since pre-trained models tend to converge faster, publicly available weights can also reduce training time, which supports small research labs that do not possess the necessary resources to pre-train large models from scratch. Shorter training times can also help to decrease the considerable energy consumption of deep learning, which has implications from an economic and environmental perspective [44].

4.5. Evaluation

The evaluation of OOD detectors is often performed by treating the original test set as IN data and testing the discriminative power against one or several different OOD datasets. Upon deployment, one would have to determine a specific threshold value τ for the discriminator in Equation (1). Using a larger threshold will reduce the recall while increasing the precision of the detector. On the other hand, a lower threshold will increase the recall while reducing the precision. There are several established metrics to measure the performance on binary classification tasks at all possible thresholds simultaneously. The AUPR measures the area under the curve obtained by plotting precision and recall against each other for all possible values of τ. Two versions of the AUPR exist: one treats OOD samples as positive, and one treats them as negative. Furthermore, performance can be measured with the AUROC, which is the area under the curve that characterizes the tradeoff between the false positive rate and the true positive rate.

Metrics PyTorch-OOD provides helper functions for the evaluation of models and also implements some OOD specific metrics that are not available in other packages at the time of writing, such as the expected calibration error described in [17], or the calculation of the accuracy at a specific true positive rate (usually 95%).

Open Set Simulations Open Set Simulations constitute an alternative evaluation protocol for OOD detector that is more commonly employed in the Open Set Recognition domain. The concept includes repeatedly

1. splitting the classes of a dataset into IN and OOD,

2. training a DNN on the IN classes, and

3. testing the OOD detectors' ability to discriminate IN from OOD classes on a test set.

A detailed description is provided in [27]. PyTorch-OOD provides a generator to create Open Set Simulations from datasets dynamically. The generator can be seeded to produce reproducible splits.

4.6. Contributing

Contributing custom implementations that extend the functionality of PyTorch-OOD is usually as easy as implementing one of several interfaces. For example, to implement a custom OOD detection algorithm, one has to implement the interface `api.Detector`, which contains the `fit` and `predict` methods described in Section 3. Training- as well as benchmark datasets, are provided in the form of torch `utils.data.Datasets`, which allows access to individual samples via the indexing operator `[]`. Objective functions and DNN architectures are implemented as torch `nn.Modules` with a `forward` methods that support automatic differentiation.

5. Experiments

In this section, we evaluate the performance of a number of methods on several benchmark tasks, including Computer Vision as well as Natural Language Processing.

5.1. CIFAR

We first present a benchmark of several models on the CIFAR 10 or CIFAR 100 dataset. As DNN, we used a WideResNet-40 and optimized its parameters by minimizing several supervised and unsupervised objective functions. Where provided, we used pre-trained weights. Furthermore, we also fine-tuned a Vision Transformer (ViT) [12] that was pre-trained on the entire ImageNet 21k dataset with standard cross-entropy. We applied a variety of different OOD detectors to both DNNs and evaluated each of them against Textures, TinyImageNet Crop, TinyImageNet Resize, LSUN Crop, and LSUN Resize. Samples from these datasets are depicted in Figure 2.

The results are depicted in Figure 4. DNNs optimized with supervision (i.e., Outlier Exposure, Energy Regularization, Objectosphere) and the ViT provide the best results. On the CIFAR 10, the performance of supervised models seems to have converged and matches or surpasses the transformer-based model's performance. However, on the more challenging CIFAR 100 dataset, the pre-trained transformer outperforms the WideResNet-40-based models, even when used with the Softmax baseline method. The performance is further increased when the transformer is combined with Energy-based OOD detection. These results confirm the general notion that high-capacity models trained with additional data can outperform more advanced methods for out-of-distribution detection.

5.2. ImageNet

In this section, we present benchmarks on the large-scale ImageNet datasets with 1000 classes [42]. As DNNs, we used a ResNet-50 [19] trained on the 1000 classes ImageNet and a ViT, pretrained on the 21k ImageNet Database [10] and subsequently fine-tuned on the ImageNet. Both models were optimized by minimizing the cross-entropy. As OOD detectors, we used the Softmax Baseline [22], as well as Energy-based OOD detection [32].

Hendrycks *et al*. propose several benchmarks for testing the robustness of ImageNet classifiers and OOD detectors. ImageNet-A [25] is a collection of images from ImageNet classes that standard DNNs frequently misclassify with high confidence. It can be used to test *input data distribution shifts*. ImageNet-O [25], on the other hand, is an out-of-distribution dataset that contains anomalous images from classes that are not part of the 1000 class ImageNet dataset. Therefore, it can be used to test *label distribution shifts*. ImageNet-R is a dataset of renditions (*e.g*., painting) of images from ImageNet classes with 30.000 images. While these images are natural, their underlying distribution diverges so much from the ImageNet distribution that most of them are misclassified. The Fooling Images dataset [36] is comprised of 10.000 synthetic images generated by an evolutionary algorithm. An example image from each dataset is provided in Figure 3.

In our evaluation, we tested the ability of the models to discriminate between the ImageNet 2012 validation set as IN data and each of ImageNet-R, ImageNet-O, ImageNet-A, and Fooling Images as OOD data, respectively. The results are depicted in Figure 5. The combination of ViT and Energy-based OOD detection again delivers the best performance across all datasets and metrics.

5.3. Newsgroups

In this section, we present results on sequential data for a Natural Language Processing task. The 20 Newsgroups dataset is a collection of 20.000 texts from 20 different online newsgroups. The task is to predict the origin newsgroup for a given document. We trained a word-level classification model based on multi-layer gated recurrent units (GRU) as used in [24]. For supervised objective functions, we used the Wikitext-2, that contains a sample of texts from the English Wikipedia, as OOD dataset.

In our evaluation, we tested the ability of the models to discriminate between the test set of the 20 Newsgroups dataset and several other text datasets that have all been used in other works for OOD detection on text data [22, 24]. The Reuters News 52 dataset includes samples of news articles

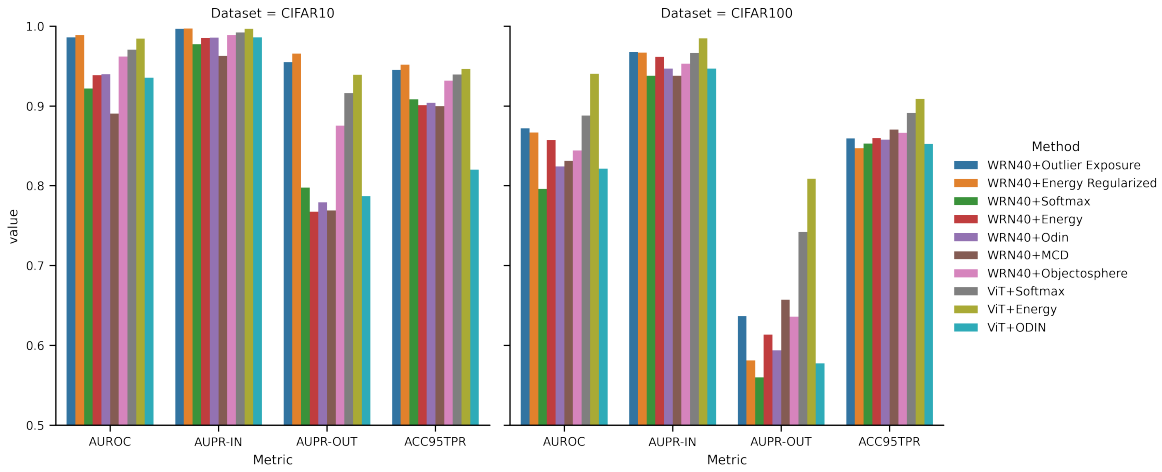

Figure 4. Scores for different OOD detection models based on a WideResNet 40 on a benchmark task including 5 OOD Datasets. Models trained with supervision (i.e., Outlier Exposure, Energy Regularization, Objectosphere) outperform unsupervised models (i.e., Softmax, Energy-Based OOD detection, Monte Carlo Dropout). On the more challenging CIFAR 100 dataset, the high capacity model based on the transformer architecture combined with the baseline method outperforms models trained with supervision.

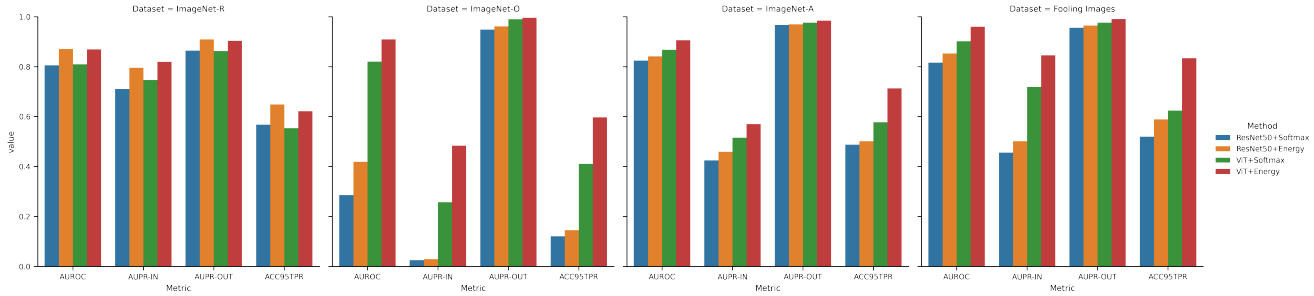

Figure 5. Scores for different OOD detection benchmark tasks for a ResNet-50 model and a Vision Transformer (ViT) [12]. For ViT+Energy, we used Energy-based OOD detection [32] with a transformer architecture, which significantly increased the performance across most metrics and datasets.

from 52 different topics. The Multi30k dataset contains english descriptions of images. The WMT16 contains example sentences from a machine translation task. Results can be found in Figure 6. We again observe that, while Energy-based OOD detection increases the performance over the baseline, supervised methods incorporating auxiliary data achieve the best results.

6. Related Work

Existing frameworks like PyOD, PyTOD, and ADTK focus on classical shallow methods for comparably low dimensional data points, time series, or streams. For tasks involving large quantities of high dimensional data, like Computer Vision and Natural Language Processing, deep models constitute the state-of-the-art, among others, due to their scalability. However, as of now, OOD detection methods for deep models are not sufficiently covered by existing libraries.

For example, PyOD [51] provides well-tested and documented implementations for a large number of classical Outlier Detection methods, such as k-Nearest Neighbors, and also some methods based on Neural Networks, like AutoEncoders. However, at the time of writing, there is minimal overlap between the detection methods provided by PyTorch-OOD and PyOD since the latter focuses more on classical methods such as those described in [1]. Neural Network architectures, pre-trained weights, and datasets, which constitute essential components required to reproduce results of the most recent publications in the OOD detection domain, are not in the scope of PyOD.

7. Conclusion & Future Work

We presented PyTorch-OOD, a library for Out-of-Distribution detection based on PyTorch. PyTorch-OOD

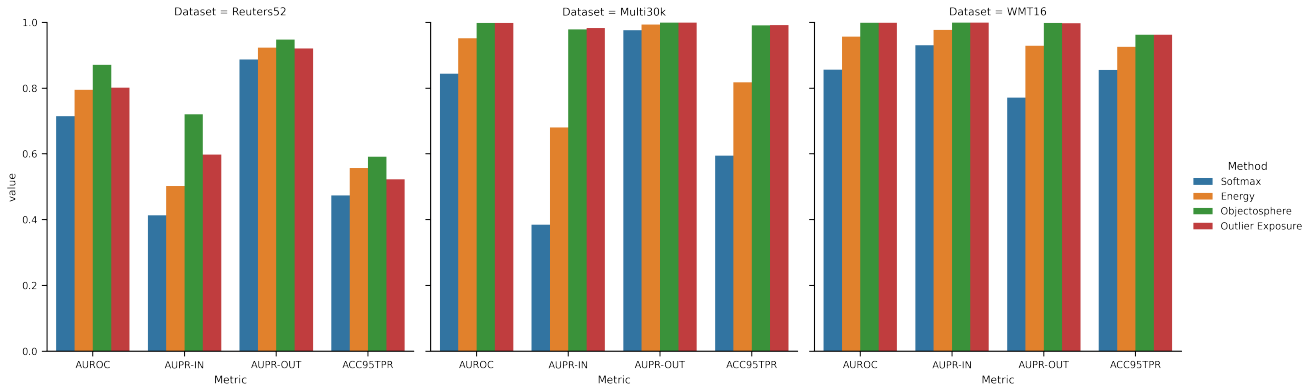

Figure 6. Scores for different OOD detection methods based on a multi-layer GRU model trained on the 20 Newsgroups dataset. For supervised objective functions, we used the Wikitext-2 dataset as OOD data. We again observe that Energy-based OOD detection outperforms the Softmax-baseline, and models trained with supervision yield the best performance.

provides implementations of several OOD detection methods and objective functions, datasets, and DNN architectures used in recent academic literature to achieve state-of-the-art results. PyTorch-OOD makes minimal assumptions on the training and evaluation procedure, which allows it to be used in conjunction with other frameworks that facilitate scalability, like PyTorch Lightning.

In the future, PyTorch-OOD can be extended by additional methods and datasets. In particular, providing easily accessible datasets can potentially facilitate OOD research on audio and video data which is, to our knowledge, not as developed as a field. We encourage others to contribute objective functions, detection methods, architectures, and datasets.

References

[1] Charu C Aggarwal. *Outlier Analysis*. Springer, 2017. 1, 7

[2] Abhijit Bendale and Terrance E Boult. Towards open set deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1563–1572, 2016. 2, 4

[3] Abeba Birhane and Vinay Uday Prabhu. Large image datasets: A pyrrhic win for computer vision? In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1536–1546. IEEE, 2021. 4

[4] Xavier Bouthillier, César Laurent, and Pascal Vincent. Unreproducible research is reproducible. In *International Conference on Machine Learning*, pages 725–734, 2019. 1, 2

[5] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013. 2

[6] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014. 2

[7] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017. 5

[8] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3606–3613, 2014. 5

[9] Robert Culkin and Sanjiv R Das. Machine learning in finance: the case of deep learning for option pricing. *Journal of Investment Management*, 15(4):92–100, 2017. 1

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 4, 6

[11] Akshay Raj Dhamija, Manuel Günther, and Terrance Boult. Reducing network agnostophobia. In *Advances in Neural Information Processing Systems*, pages 9157–9168, 2018. 3

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *International Conference on Learning Representations*, 2021. 6, 7

[13] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24–29, 2019. 1

[14] Stanislav Fort, Jie Ren, and Balaji Lakshminarayanan. Exploring the limits of out-of-distribution detection. *Advances in Neural Information Processing Systems*, 34, 2021. 5

[15] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learn-

ing. In *International Conference on Machine Learning*, pages 1050–1059. PMLR, 2016. 1, 4

[16] Chuanxing Geng, Sheng-jun Huang, and Songcan Chen. Recent advances in open set recognition: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2020. 1

[17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR, 2017. 5

[18] Mehadi Hassen and Philip K Chan. Learning a neural-network-based representation for open set recognition. In *Proceedings of the 2020 SIAM International Conference on Data Mining*, pages 154–162. SIAM, 2020. 3

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 6

[20] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, et al. The many faces of robustness: A critical analysis of out-of-distribution generalization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8340–8349, 2021. 5

[21] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*, 2019. 4, 5

[22] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *International Conference on Learning Representations*, 2017. 4, 5, 6

[23] Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. In *International Conference on Machine Learning*, pages 2712–2721. PMLR, 2019. 5

[24] Dan Hendrycks, Mantas Mazeika, and Thomas Dietterich. Deep anomaly detection with outlier exposure. In *International Conference on Learning Representations*, 2018. 3, 4, 5, 6

[25] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021. 5, 6

[26] Young Jin Heo, Dayeon Kim, Woongyong Lee, Hyoungkyun Kim, Jonghoon Park, and Wan Kyun Chung. Collision detection for industrial collaborative robots: A deep learning approach. *IEEE Robotics and Automation Letters*, 4(2):740–746, 2019. 1

[27] Konstantin Kirchheim, Tim Gonschorek, and Frank Ortmeier. Addressing randomness in evaluation protocols for out-of-distribution detection. *2nd Workshop on Artificial Intelligence for Anomalies and Novelties at IJCAI*, 2021. 1, 6

[28] Rajat Koner, Poulami Sinhamahapatra, Karsten Roscher, Stephan Günnemann, and Volker Tresp. OODformer: Out-of-distribution detection transformer. In *British Machine Vision Conference*, 2021. 5

[29] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. 1

[30] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in Neural Information Processing Systems*, 31, 2018. 2, 4, 5

[31] Shiyu Liang, Yixuan Li, and R Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. In *6th International Conference on Learning Representations, ICLR 2018*, 2018. 4, 5

[32] Weitang Liu, Xiaoyun Wang, John Owens, and Yixuan Li. Energy-based out-of-distribution detection. *Advances in Neural Information Processing Systems*, 33, 2020. 3, 4, 5, 6, 7

[33] Dimity Miller, Niko Sunderhauf, Michael Milford, and Feras Dayoub. Class anchor clustering: A loss for distance-based open set recognition. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3570–3578, 2021. 3

[34] Norman Mu and Justin Gilmer. MNIST-C: A robustness benchmark for computer vision. In *ICML Workshop on Uncertainty and Robustness in Deep Learning*, 2019. 5

[35] Prabhat Nagarajan, Garrett Warnell, and Peter Stone. The impact of nondeterminism on reproducibility in deep reinforcement learning. In *2nd Reproducibility in Machine Learning Workshop at ICML*, 2018. 1

[36] Anh Nguyen, Jason Yosinski, and Jeff 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*, pages 427–436, 2015. 4, 5, 6

[37] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019. 1

[38] Stanislav Pidhorskyi, Ranya Almohsen, Donald A Adjeroh, and Gianfranco Doretto. Generative probabilistic novelty detection with adversarial autoencoders. *Advances in Neural Information Processing Systems*, 31, 2018. 1

[39] Lukas Ruff, Jacob R Kauffmann, Robert A Vandermeulen, Grégoire Montavon, Wojciech Samek, Marius Kloft, Thomas G Dietterich, and Klaus-Robert Müller. A unifying review of deep and shallow anomaly detection. *Proceedings of the IEEE*, 2021. 1

[40] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *International conference on machine learning*, pages 4393–4402. PMLR, 2018. 4

[41] Lukas Ruff, RA Vandermeulen, N Gornitz, Alexander Binder, E Muller, and Marius Kloft. Deep support vector data description for unsupervised and semi-supervised anomaly detection. In *Proceedings of the ICML 2019 Workshop on Uncertainty and Robustness in Deep Learning, Long Beach, CA, USA*, pages 9–15, 2019. 3

[42] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015. 6

[43] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015. 1

[44] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, 2019. 5

[45] Cecilia Summers and Michael J. Dinneen. Nondeterminism and instability in neural network optimization. In *International Conference on Machine Learning*, pages 9913–9922. PMLR, 2021. 1, 2

[46] Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008. 4, 5

[47] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In *European conference on computer vision*, pages 499–515. Springer, 2016. 3

[48] Jingkang Yang, Kaiyang Zhou, Yixuan Li, and Ziwei Liu. Generalized out-of-distribution detection: A survey. *ArXiv*, abs/2110.11334, 2021. 4

[49] Harish Yenala, Ashish Jhanwar, Manoj K Chinnakotla, and Jay Goyal. Deep learning for detecting inappropriate content in text. *International Journal of Data Science and Analytics*, 6(4):273–286, 2018. 1

[50] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference 2016*. British Machine Vision Association, 2016. 5

[51] Yue Zhao, Zain Nasrullah, and Zheng Li. PyOD: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019. 7