Presentation of OODEEL
Paul Novello, DEEL, IRT Saint Exupery
Outline

1. Introduction: Post-hoc OOD detection
   ➢ Model based vs Model agnostic
   ➢ Post-hoc vs training
   ➢ Some examples

2. Existing OOD detection libraries and positioning of OODEEL

3. OODEEL in practice
Post-hoc OOD detection

Two approaches to OOD detection:

• Model agnostic
  
  **Goal:** find data that does not belong to the same distribution as some input data 
  **Application:** Anomaly detection, Outlier detection

• Model based
  
  **Goal:** find data that does not belong to the same distribution as some input data *that is used to train a model for some auxiliary ML task (classification, object detection)* 
  **Application:** Robustness, selective inference, model monitoring
Post-hoc OOD detection

Model based

**Goal:** find data that does not belong to the same distribution as some input data *that is used to train a model for some auxiliary ML task (classification, object detection)*

**Application:** Robustness, selective inference, monitoring

OOD Detection by design

- **Unseen data**
  - OOD aware training procedure
  - model capable of OOD detection
  - Inference
  - Prediction
  - Keep prediction?

**Advantage:**
- OOD detection by design, the model can be used as is

**Drawback:**
- Needs a whole new and specific training procedure
Post-hoc OOD detection

Model based

Goal: find data that does not belong to the same distribution as some input data *that is used to train a model for some auxiliary ML task (classification, object detection)*

Application: Robustness, selective inference, monitoring

Post-hoc OOD Detection

- Unseen data
- Any already trained model
- Inference
- Post-hoc OOD algorithm
- Keep prediction?
Post-hoc OOD detection: Example 1 - DKNN

Penultimate Layer’s Feature

The k-th Nearest Neighbor Distance Distribution

Out-of-Distribution Detection with Deep Nearest Neighbors, Sun et al., ICML 2022
Post-hoc OOD detection: Example 2 - VIM

ViM: Out-Of-Distribution with Virtual-logit Matching, Wang et al. CVPR 2022
Post-hoc OOD detection

Model based

Goal: find data that does not belong to the same distribution as some input data that is used to train a model for some auxiliary ML task (classification, object detection)

Application: Robustness, selective inference, monitoring

Post-hoc OOD Detection

Unseen data ——> Any already trained model ——> Inference ——> Prediction

Advantage:
• Can be applied to any pretrained model

Drawback:
• Less accurate OOD performances ???

e.g. hugging face, keras.applications…
Is Post-hoc OOD Detection really less accurate?

"Post-Hoc Methods Outperform Training in General"

For OOD and OSR methods without extra data, we further split them into two parts, one that needs a training process and the other does not. Surprisingly, those methods that require training do not necessarily obtain higher performance. Generally, methods that require training do not outperform inference-only methods. Nevertheless, the trained models can be generally used in a combined way with the post-hoc methods, which could potentially further increase their performance.


... not so clear

1. Post-hoc methods are at least as good as training based
2. They are way more convenient because they do not require any training and can be applied to already trained networks

Table 1: Experimental Results on the Generalized OOD Detection Benchmark. The generalized OOD detection benchmark composes 9 benchmarks that are popular in the subfields of AD, ODR, and OOD detection. To save space, we use M-6 to denote MNIST-64, C-6 to CIFAR-64, C-50 to CIFAR-50/50, TIN-20 to TinyImageNet20/10 benchmark. We only report the mean of AUCROC.
Outline

1. Introduction: Post-hoc OOD detection
2. Existing OOD detection libraries and positioning of OODEEL
   - OpenOOD
   - PyOD
   - Pytorch-OOD
3. OODEEL in practice
Existing OOD libraries and positioning of OODEEL

OpenOOD

- Focused on deep methods
- Extensive, encompasses all the existing deep approaches, and many algorithms
- As a result, less flexible, coherent and more difficult to use.
- More of a benchmarking software than a library
- Pytorch Only

[link to OpenOOD GitHub repository]

This repository reproduces representative methods within the Generalized Out-of-Distribution Detection Framework, aiming to make a fair comparison across methods that initially developed for anomaly detection, novelty detection, open set recognition, and out-of-distribution detection. This codebase is still under construction. Comments, issues, contributions, and collaborations are all welcomed!

Timeline of the methods that OpenOOD supports.
Existing OOD libraries and positioning of OODEEL

PyOD

yzhao062/pyod: A Comprehensive and Scalable Python Library for Outlier Detection (Anomaly Detection) (github.com)

- Very coherent and easy to use (see example)
- Broader than deep learning based methods
- As a result, lacks deep training based and post-hoc methods

PyOD is featured for:

- Unified APIs, detailed documentation, and interactive examples across various algorithms.
- Advanced models, including classical distance and density estimation, latest deep learning methods, and emerging algorithms like ECOD.
- Optimized performance with JIT and parallelization using numba and joblib.
- Fast training & prediction with SUOD [46].

Outlier Detection with 5 Lines of Code:

```python
# train an ECOD detector
from pyod.models.ecod import ECOD
clf = ECOD()
clf.fit(X_train)

# get outlier scores
y_train_scores = clf.decision_scores_  # raw outlier scores on the train data
y_test_scores = clf.decision_function(X_test)  # predict raw outlier scores on test
```
Existing OOD libraries and positioning of OODEEL

PytorchOOD

GitHub - kkirchheim/pytorch-ood: PyTorch Out-of-Distribution Detection

- Very coherent and easy to use (see example)
- Focus on deep training based and post-hoc methods
- Pytorch Only


The library provides:

- Out-of-Distribution Detection Methods
- Loss Functions
- Datasets
- Neural Network Architectures as well as pretrained weights
- Useful Utilities

and is designed such that it should be compatible with frameworks like pytorch-lightning and pytorch-segmentation-models. The library also covers some methods from closely related fields such as Open-Set Recognition, Novelty Detection, Confidence Estimation and Anomaly Detection.

```python
from pytorch_ood.model import WideResNet
from pytorch_ood.detector import EnergyBased
from pytorch_ood.utils import OODMetrics

# Create Neural Network
model = WideResNet(pretrained="er-cifar10-tune").eval().cuda()

# Create detector
detector = EnergyBased(model)

# Evaluate
metrics = OODMetrics()

for x, y in data_loader:
    metrics.update(detector(x,x.cuda()), y)

print(metrics.compute())
```
## Existing OOD libraries and positioning of OODEEL

<table>
<thead>
<tr>
<th>Lib</th>
<th>Post-hoc</th>
<th>Training</th>
<th>Simple API</th>
<th>tensorflow</th>
<th>pytorch</th>
</tr>
</thead>
<tbody>
<tr>
<td>PyOD (not deep)</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>OpenOOD</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Pytorch-OOD</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>OODEEL</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
Outline

1. Introduction: Post-hoc OOD detection
2. Existing OOD detection libraries and positioning of OODEEL
3. OODEEL in practice
   - Benchmarking methodologies
   - Dataset wise tuto: MNIST vs Fashion MNIST
   - Class wise tuto: MNIST [0-4] vs MNIST [5-9]
   - Elecboard Components
   - Eurosat
   - Going further: LARD
Test case: benchmarking

Two common ways of benchmarking baselines:

1. **Dataset wise**: Consider one dataset as ID, and another as OOD
2. **Class wise**: For a classification dataset with $n$ classes, consider $k$ classes of as ID and $n - k$ classes as OOD

ID: MNIST

OOD: CIFAR
Test case: benchmarking

Two common ways of benchmarking baselines:

1. Dataset wise: Consider one dataset as ID, and another as OOD
2. Class wise: For a classification dataset with $n$ classes, consider $k$ classes of as ID and $n - k$ classes as OOD

ID: MNIST = 5  
OOD: MNIST ≠ 5
**OODEEL in practice – Elecboard Components**

**Elecboard Components**

**Dataset DEEL**

- 10,000 images of electronic components on circuit boards (64x64x3, RGB)
- 27 classes

**Experiment:** OOD detection to discover unknown components

Samples of **Elecboard Components dataset**.

We arbitrary choose 20 ID and 7 OOD classes for our experiment.
OODEEL in practice – Elecboard Components

ID / OOD data
- **ID (train distribution):** 20 components classes with similar shape, size and color
- **OOD:** 7 other components classes

Model
- ResNet20 (suited for small images)
- Trained on **ID data** (97.8% of test accuracy)
- No data augmentation
- Pretrained on CIFAR-10
OODEEL in practice – Elecboard Components

OOD Scores

Histograms of the OOD scores for ID and OOD data (Elecboard Component test set)

<table>
<thead>
<tr>
<th></th>
<th>DKNN</th>
<th>VIM</th>
<th>Energy</th>
<th>ODIN</th>
<th>Mahalanobis</th>
<th>MLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUROC</td>
<td>0.9999</td>
<td>0.992</td>
<td>0.345</td>
<td>0.538</td>
<td>1.0</td>
<td>0.356</td>
</tr>
<tr>
<td>FPR95</td>
<td>0.0008</td>
<td>0.042</td>
<td>0.954</td>
<td>0.919</td>
<td>0.0</td>
<td>0.948</td>
</tr>
</tbody>
</table>

AUROC and FPR95 metrics to evaluate OOD detectors
OODEEL in practice – EuroSat dataset

EuroSat

https://github.com/phelber/EuroSAT

- 27,000 satellite images of shape 64x64x3 (Sentinel 2, RGB)
- 10 classes

A few samples on EuroSat dataset.
OODEEL in practice – EuroSat dataset

First scenario: OOD detection to discover new types of land

ID / OOD data
- **ID (train distribution):** Rural classes (Forest, Herbaceous vegetation, River etc.)
- **Near OOD:** Slightly urbanized class (Highway)
- **Far OOD:** Strongly urbanized classes (Industrial, Residential)

Model
- **ResNet20** (suited for small images)
- Trained on **Rural classes** (97.8% of test accuracy)
- No data augmentation
- Pretrained on CIFAR-10
ODEEL in practice – EuroSat dataset

OOD scores

Near OOD: Highway

Far OOD: Industrial, Residential

Histograms of the OOD scores for ID vs OOD data

<table>
<thead>
<tr>
<th></th>
<th>MLS</th>
<th>VIM</th>
<th>DKNN</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUROC</td>
<td>0.872</td>
<td>0.795</td>
<td>0.918</td>
<td>0.872</td>
</tr>
<tr>
<td>FPR95</td>
<td>0.511</td>
<td>0.536</td>
<td>0.296</td>
<td>0.511</td>
</tr>
</tbody>
</table>

Histograms of the OOD scores for ID vs OOD data

<table>
<thead>
<tr>
<th></th>
<th>MLS</th>
<th>VIM</th>
<th>DKNN</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUROC</td>
<td>0.906</td>
<td>0.850</td>
<td>0.926</td>
<td>0.906</td>
</tr>
<tr>
<td>FPR95</td>
<td>0.331</td>
<td>0.409</td>
<td>0.263</td>
<td>0.331</td>
</tr>
</tbody>
</table>

For GOAD, ICLR2020, an Anomaly Detection method:
0.73 AUROC while involving another training
OODEEL in practice – EuroSat dataset

Second scenario: OOD score as a proxy for classification error

Model
• ResNet20 (suited for small images)
• Trained on the whole dataset (97.6% of test accuracy)
• No data augmentation
• Pretrained on CIFAR-10

With correctly predicted samples taken as ID data and incorrectly predicted samples taken as OOD data:

<table>
<thead>
<tr>
<th>VIM</th>
<th>AUROC (↑)</th>
<th>TPR5FPR (↓)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.893</td>
<td>0.349</td>
</tr>
</tbody>
</table>

=> Means that if we raise alarms for 5% of workable data (on which the classifier is accurate), we avoid 34.9% of the remaining errors
OODEEL in practice – Going further with LARD
Reminder: FCOS architecture
What is OOD in object detection
Back to bounding boxes embeddings
Using OODEEL for OOD in object detection
Using OODEEL for OOD in object detection

Custom class inherited from built in with only few method overrides

```python
extractor = FcosFeatureExtractor(model)
ood = DN3D()
ood.fit(extractor, data_loader_fit, backend="torch")
scores_calib, _ = ood.score(data_loader_test)
```
OOD in object detection: OOD bbox filtering
Conclusion and Takeaway

- **Post-hoc** OOD detection is a convenient and efficient way of performing OOD detection on already trained models.
Conclusion and Takeaway

- Post-hoc OOD detection is a convenient and efficient way of performing OOD detection on already trained models.
- OODEEL implements various recent Post-Hoc OOD detection methods.
Conclusion and Takeaway

- Post-hoc OOD detection is a convenient and efficient way of performing OOD detection on already trained models
- OODEEL implements various recent Post-Hoc OOD detection methods
- OODEEL works for pytorch and tensorflow already-trained models
Conclusion and Takeaway

- Post-hoc OOD detection is a convenient and efficient way of performing OOD detection on already trained models.
- OODEEL implements various recent Post-Hoc OOD detection methods.
- OODEEL works for pytorch and tensorflow already-trained models.
- Many development efforts have been put into making OODEEL simple to use for various test cases (dataset wise and class wise benchmarking).
Conclusion and Takeaway

- Post-hoc OOD detection is a convenient and efficient way of performing OOD detection on already trained models.
- OODEEL implements various recent Post-Hoc OOD detection methods.
- OODEEL works for PyTorch and TensorFlow already-trained models.
- Many development efforts have been put into making OODEEL simple to use for various test cases (dataset wise and class wise benchmarking).
- Specifically, it is very easy to customize OODEEL and to add your own method (not covered in the presentation but see this doc tutorial).
Conclusion and Takeaway

- Post-hoc OOD detection is a convenient and efficient way of performing OOD detection on already trained models.
- OODEEL implements various recent Post-Hoc OOD detection methods.
- OODEEL works for PyTorch and TensorFlow already-trained models.
- Many development efforts have been put into making OODEEL simple to use for various test cases (dataset wise and class wise benchmarking).
- Specifically, it is very easy to customize OODEEL and to add your own method (not covered in the presentation but see this doc tutorial).
- OODEEL is a library ready for real world applications but it is also very adapted for research (e.g. for object detection or benchmarking new OOD methods).
Conclusion and Takeaway

- Post-hoc OOD detection is a convenient and efficient way of performing OOD detection on already trained models.
- OODEEL implements various recent Post-Hoc OOD detection methods.
- OODEEL works for PyTorch and TensorFlow already-trained models.
- Many development efforts have been put into making OODEEL simple to use for various test cases (dataset wise and class wise benchmarking).
- Specifically, it is very easy to customize OODEEL and to add your own method (not covered in the presentation but see this doc tutorial).
- OODEEL is a library ready for real world applications but it is also very adapted for research (e.g. for object detection or benchmarking new OOD methods).
What to do next?

A post-hoc deep OOD detection toolbox
Thank you for your attention