GMDL222 – Final Project

Oren Freifeld and Ron Shapira Weber
The Department of Computer Science, Ben-Gurion University

Release Date: 01/06/2022
Submission Deadline: 30/07/2022 , 23:59

Abstract

Our final project focuses on the problem of Open Set Recognition (OSR). Unlike traditional classification problems, where a model is trained and tested on the same set of classes, OSR refers to a scenario where, in addition to the known classes the classifier was trained on, additional new classes might be present during test-time. In fact, this is closely related to the problem of out-of-distribution (OOD) detection.

More concretely, in this project, you are tasked with an image classification problem on the MNIST dataset. However, your model should be able to not only correctly classify MNIST examples but also flag unseen classes during test-time as 'Unknown'. This means, for example, that during test you will get not only examples from MNIST's test dataset but also examples from possibly unrelated datasets (*e.g.*, you may get letters from some alphabet or even images of animals, *etc.*).

Contents

Version Log

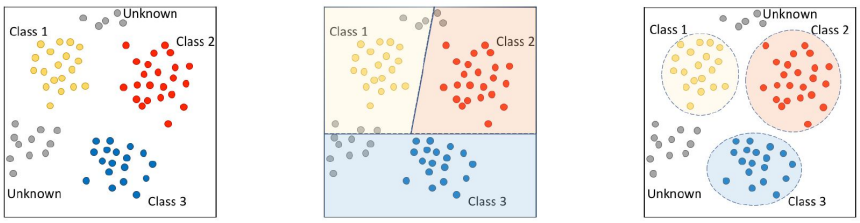
- 1.00, 01/06/2022. Initial release.

1 Preliminaries

Read before you start:

- Use PyTorch to build and train your model.
- You are required to submit a Colab notebook.
- As you have seen in the practical session, a convenient way to access and download the MNIST dataset is via `torchvision.dataset.MNIST` (see DL PS1 for more details).
- Optional (but strongly recommended) reading to familiarize yourself with several OSR approaches: Recent Advances in Open Set Recognition: A Survey [1]

2 Open Set Recognition



(a) (b) (c)

Figure 1: The Open Set Recognition problem. (a) Data distribution, (b) Traditional classification problem and (c) OSR problem.

A good description of the OSR problem is given in [1]:
"In real-world recognition/classification tasks, limited by various objective factors, it is usually difficult to collect training samples to exhaust all classes when training a recognizer or classifier. A more realistic scenario is open set recognition (OSR),where incomplete knowledge of the world exists at training time, and unknown classes can be submitted to an algorithm during testing, requiring the classifiers to not only accurately classify the seen classes, but also effectively deal with unseen ones."

You will be evaluated on the following:

- Correctly classifying the 10 digits of the MNIST dataset.
- Correctly classifying unseen classes as "unknown'.

As such, there are several requirements you should try and meet:

1. Mitigate the drop in accuracy w.r.t your baseline model as much as possible.
2. Limit the computational overhead of your OSR classifier.

3. Being able to distinguish between low-confidence samples such as poorly-written digit and an out-of-distribution ones, such as an image of a bird.
4. Correctly classify unseen classes as 'Unknown'.

Given these requirements, you have a relativity free hand in choosing your OSR strategies. Here are a few guidelines and hints that should get you started:

- **An additional neuron for the 'unknown' class**: That being said, please see the 'methods which are not allowed' for the limitation of such methods.
- **The embedding layer**: usually referring to the penultimate layer of a classification model, the embedding layer represents a data point as a vector in the 'embedded space' which should have a 'good' separation between classes for the SoftMax layer/classifier to properly distinguish between. As seen in Fig 1, in the OSR setting we might want to properly model the embedded distribution rather then draw decisions boundaries. Therefore, instead of adding another neuron of the 'unknown' class prediction, you might want to model the 10 MNIST classes in the embedding space instead. Training a clustering algorithm on the embedded space is allowed.
- **Uncertainty-based methods**: given a **calibrated** model which outputs proper class probabilities, it is possible to leverage the uncertainty in the model's prediction to flag unseen classes. Monte-Carlo Dropout, (small) ensembles and test-time augmentations are allowed.
- **Data augmentation**: highly recommended, both for increasing the models generalization ability and increasing the 'size' of the 'closed set'. Meaning, augmented images such as blurry/rotated/low-res ones are considered in-distribution (though, you should still be able to tell your sixes from your nines).
- **Autoencoders (AE)**: AE compress their input into a 'encoded' vector given a reconstruction error. They could be leveraged in two ways for the OSR problem: (1) The 'encoded' vector could be utilized to model the known classes via clustering algorithm. (2) The reconstruction error could be monitored for unseen classes (i.e., new classes might invoke relativity high reconstruction error).

Methods which are **NOT** allowed:

- Training with additional class/es: Several methods train their model with $K+1$ classes, where the 'unknown' classes is taken from a different distribution. This is a BAD solution when working with real-world data, since you don't know from which distribution(s) your unseen classes might come from.

3 Data

As mentioned earlier, this project will use MNIST as the "closed set" on which you model will be trained on. `torchvision.dataset.MNIST` holds the train-test sets as PyTorch datasets. However, you should create a validation set and dataloaders on your own. **You are NOT allowed to <u>train</u> your model on any dataset other than the MNIST dataset for this project.** Use CIFAR10 as a placeholder for OOD data for you to evaluate your model.

Remark 1 *Note well: this does* ***not*** *mean that the test data that we will use will come from CIFAR10.* ◊

Mandatory Transformations: while you are not required to use data augmentation, you are encouraged to do so. Regardless of data augmentation, there are a few mandatory operations which you are required to implement as a part of your data processing pipeline:

- `ToTensor()`.
- `Normalize()` using the train set mean and standard deviation.
- `Resize()` to 28x28.

4 Model

Your model should be fairly similar to the ones specified at HW5/HW6. While larger models might get you an increase of 2-5% test accuracy, for the purposes of this project a simple CNN should suffice (and will train much faster). Thus, your focus should be on OSR strategies rather then the #parameters of your model.

- Your model should consist of 2–3 Conv layers and 1–2 fully-connected (FC) layers. You are free to use whatever activation functions you see fit. You may also add batch-normalization layers, dropout, pooling and residual connections, and change the size of the filters.
- Pay special attention to the embedding layer: many OSR methods utilize it for the task. In addition, if you have decided not to use average-pooling, the FC layers might hold the vast majority of the trainable parameters of your model so be mindful of its size.
- You may add another neuron to the last layer (and SoftMax) for the 'Unknown' class prediction (that said, you may choose to build an OSR solution that is not based on an additional such neuron).
- You may choose an Autoencoder (AE) instead of (or in conjunction with) the proposed CNN. If so, it should be roughly similar to the one introduced in HW6. Remember, you should also be able to correctly classify the MNIST digits, not just to flag the OOD data. You may want to consider using a multi-task AE:
  - Domain Generalization for Object Recognition with Multi-task Autoencoders (ECCV '15)
  - PyTorch code example (GitHub)

5 Evaluation

You tasks are the following:

Computer Exercise 1 *Compute the baseline accuracy of your model on the MNIST test set (*i.e.*, the baseline of your OSR model).* ◊

Computer Exercise 2 *Build an OSR model which predict if a data sample is one of the 10 MNIST classes or an unseen class. Have a dictionary which maps the output of your model from* $\{0, ..., 10\}$ *to* $\{0, ..., 9, 'Unknown'\}$ *(i.e., 10='Unknown').* ◊

The OSR test set for this task is stored under lock and key and will be used to evaluate your OSR model after submission. That said, the MNIST test set is available to you and should be used to evaluate you baseline model. For evaluation of OSR data, please do the following:

1. Use the CIFAR10 dataset as an OSR dataset (100-500 samples should be enough). Make sure to convert each image to grayscale and resize accordingly.
2. Create a designated pytorch dataset and a dataloader for the evaluation. All labels should be set to 10 ('Unknown')

5.1 Project Outline

Your submission should follow the project outline detailed below. Every subsection should be accompanied by a description in the form of code comments and/or markdowns.

5.1.1 Data & Preprocessing

- Datasets, dataloaders and transformations.
- Train, validation and test splits
- A subset of CIFAR10 as OOD dataset.
- Random seeds (for reproducibility; optional).

5.1.2 Models

- Baseline model class
- OSR model class

5.1.3 Training

- Training procedure for both models.

5.1.4 Evaluation

- Baseline results: accuracy and the Confusion matrix of your model performance on the MNIST test set.
- OSR rational: describe your method and the rational behind it. Cite the relevant paper(s) if needed. If this is not your first attempt, you are encouraged to describe previous attempts and what you have learned from them.
- OOD results: binary classification accuracy and a Confusion matrix for the new data. Two classes: 'Known' and 'Unknown'. Simply map the predictions of all 10 classes of MNIST to 0 and OSR to 1. No need for dedicated training.

- OSR results: evaluate the total accuracy on both the 10 MNIST classes and the unseen data. Plot a confusion matrix for the 11 classes (10+Unknown).
- Discuss how you predict your model will perform given other OOD data.
- (optional) t-SNE visualization applied to the embedded layer of the test data (color each class according to the prediction; there should be 11 colors in total).
- Remember: CIFAR10 is simply a placeholder for the true OSR data. Good performance on CIFAR does not indicate good performance on the true OSR set.

6 Submission Instructions

Provide a link to Colab notebook that you shared with ronsha@post.bgu.ac.il. In addition, please provide a .zip file containing:

- trained models weights*.
- a PDF file of the already-run notebook.

Do not forget to follow the general HW instructions from the course Moodle.

*One way to download files from Colab:

```
from google.colab import files
files.download('examplefile.txt')
```

To load weights, simply uploaded them to your project's drive folder and load from colab (similarly to how we loaded ants and bees images).