GMDL - Final Project

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

Release Date: 27/06/2024
Submission Deadline: 15/08/2024, 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

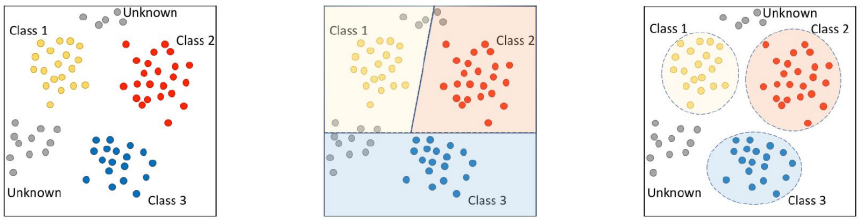
- 1.0.0, 27/06/2024. Initial release.

1 Preliminaries

Read before you start:

- You are required to submit an already-run notebook.
- You are required to submit an addition PDF with your answer and figures.
- You are required to use the `project_utils.py` file.
- Optional (but strongly recommended) reading to familiarize yourself with several OSR approaches: Recent Advances in Open Set Recognition: A Survey

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 pipeline - an overall of around 10 minutes max for the entire notebook to run.

3. Being able to distinguish between low-confidence samples such as poorly-written digits and 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**: increasing the number of classes predicted by the SoftMax layer to $K + 1$. 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 Figure 1, in the OSR setting we might want to properly model the embedded distribution rather than draw decision 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 (i.e., training a clustering algorithm on the embedded points).
- **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. For instance, Monte-Carlo Dropout, (small) ensembles and test-time augmentations.
- **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 an 'encoded' vector given a reconstruction error. They could be leveraged in a few ways for the OSR problem: (1) The 'encoded' vector could be utilized to model the known classes via a clustering algorithm. (2) The reconstruction error could be monitored for unseen classes. (3) Multi-task AEs.

Methods which are **NOT allowed**: Training with additional classes/es: Several methods train their model with $K + 1$ classes, where the 'unknown' classes are 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.

3 Data

As mentioned earlier, this project will use MNIST as the "closed set" on which your 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 and/or FashionMNIST 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/FashionMNIST.*

Data Example:

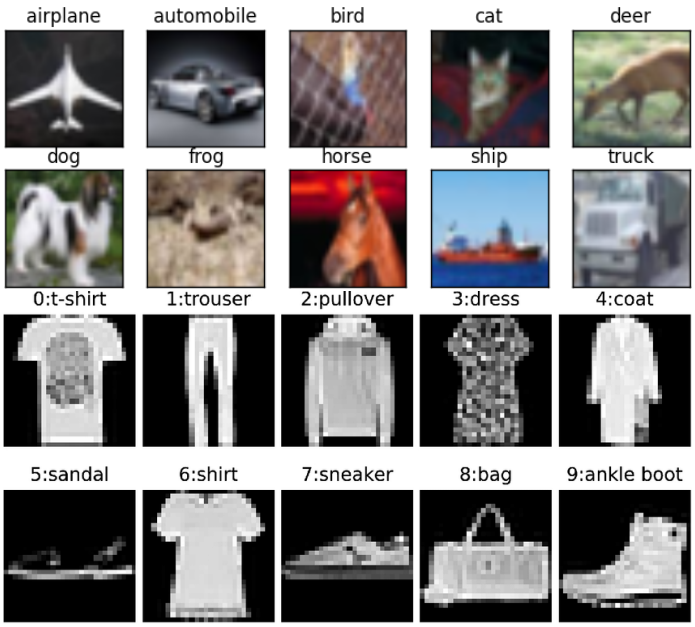


Figure 2: Top: CIFAR10. Bottom: FashionMNIST.

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 that 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 28×28.

4 Evaluation

Your 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 predicts if a data sample is one of the 10 MNIST classes or an unseen class. Have a dictionary that*

maps the output of your model from $\{0, \ldots, 10\}$ *to* $\{0, \ldots, \text{'Unknown'}\}$ *(i.e.,* $10 =$ *'Unknown').* $\Diamond$

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 your baseline model. For evaluation of OSR data, please do the following:

1. Use the `CIFAR10/FashionMNIST` datasets as an OSR dataset (1000-2000 samples should be enough). Make sure to convert each image to grayscale and resize accordingly.
2. Create a designated `CIFAR10/FashionMNIST` pytorch dataset and a dataloader for the evaluation. All labels should be set to 10 ('Unknown')
3. Create a designated `OSR` pytorch dataset and a dataloader for the evaluation. It should merge the MNIST test set and CIFAR10/FashionMNIST test samples. This dataset and dataloader will be replaced by the true OSR dataset when evaluating your project, make sure it is clear and easy to use.
4. Use the functions from `project_utils.py` for your OSR dataset and evaluation function.

4.1 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.

4.1.1 General

- Names and IDs
- Import relevant packages
- Create a boolean named `is_training` to indicate whether the entire notebook will be used for training or only evaluation. If `is_training == True`, then pressing 'Runtime' → 'Run all' on Colab should not perform any training/fitting of your models. Instead, it should load the relevant weights and only perform evaluation.

4.1.2 Data & Preprocessing

- Datasets, dataloaders, and transformations.
- A subset of CIFAR10/FashionMNIST as OOD dataset.
- Random seeds (for reproducibility; optional)

4.1.3 Models

- Baseline model class
- OSR model class

4.1.4 Training

- Training procedure for both models.

4.1.5 Evaluation

1. Baseline results: accuracy and the confusion matrix of your model performance on the MNIST test set.
2. Describe your OSR approach.
3. 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.
4. 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).
5. (optional) t-SNE and/or PCA visualization applied to the embedded layer of the test data (color each class according to the prediction; there should be 11 colors in total).
6. Additional figures (optional): loss/accuracy during training, ROC Curves, relevant distribution, model overview, *etc.*
7. Remember: CIFAR10/FashionMNIST is simply a placeholder for the true OSR data. Good performance on CIFAR does not indicate good performance on the true OSR set.

4.2 Grading

The general composition of the grade is 80% project quality and 20% performance. The performance portion of the grade (20%) will be the overall accuracy of your OSR model. This includes both correctly classifying MNIST digits and the Unknown sample.

The remaining 80% will be given according to the general quality of your project (code and summary). For instance:

- Code - should be clear with documentation
- OSR approach
- Efficiency
- Analysis of results: claims should be backed with metrics/figures
- Presentations: plot axes should have labels, legends, and titles. Images should be clear. No redundant prints (A single plot of the training procedure is worth a thousand `"print(accuracy, epoch)"`).

4.3 Project Summary

Finally, you are required to submit an additional .pdf file summarizing your project:

1. 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.
2. What hyper-parameters does your method have? How and why did you pick their final configuration?
3. Attach relevant figures from your project (current and past attempts). How does these figures indicate your performance?
4. Limitation - what are the limitations of your approach? On what types of data should it perform well/poorly?

5 Submission Instructions

Provide a .zip file containing:

- Trained models weights*
- A project summary (pdf file)
- The already-run notebook (.ipynb file)
- Additional scripts if needed

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')
```

Another way is via the file explorer, right-click and download.