Learning Data Placeholder for Open Set Recognition

Indian Statistical Institute, Kolkata

ECSU Summer School,2023

Soumyadweep Mondal

Supervisor : Aditya Panda

14/07/2023

Abstract

Closed set classifier models are trained on images from a set of classes and the test set consists of images from the same classes. The existing deep learning based classifiers are very good at classifying images from these known classes. But, if an image comes from outside of the trained classes[2] then the model may over-confidently predict it as a known image. This problem is formally known as **Open-Set Recognition** problem. In this work we propose a method to handle Open Set Recognition problem. The existing works depends on calibration and thresholding. Calibration involves setting a threshold for the probability of a prediction. If the probability is below the said threshold then the model will predict as novel class. But calibration based methods suffer from overfitting problem. We propose to address this issue by allocating **placeholders** for images of different classes from the dataset. The **data placeholders** are used to anticipate the feature distribution for novel classes. This is done by randomly mixing image features from known classes. Hence converting closed set training to open set training. This helps the model to learn to learn robust decision boundaries between known and novel classes. Our proposed approach is validated on MNIST dataset.

1. Introduction

It is not possible to train all novel class categories. In this scenario the model predicts all novel classes as known one. Hence, the performance of the model decays, making it unacceptable in real world scenarios.[2,4]

one way to seperate known and unknown classes is putting a threshold on the output probability. The known instances with higher probability and unknown instances with lower probability. But deep learning models may prone to overfitting. As a result, the model would output a high probability even for an unknown class instance, making the threshold hard to tune.

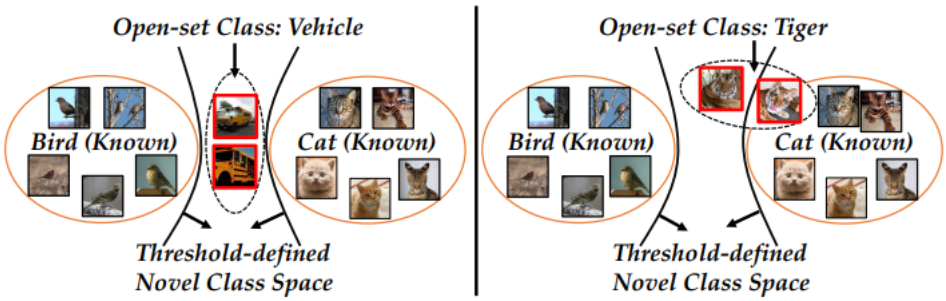

Figure 1: Since the distributions of the vehicle and tiger categories differ, it is hard to rely on a single threshold to recognize unknown classes with diverse characteristics. The same threshold which well separates vehicles apart from known classes is not suitable for tigers

There are a number of reasons why deep learning models can be prone to overfitting. One reason is that deep learning models have a large number of parameters. This means that they have the potential to learn very complex patterns in the training data. However, if the training data is not large enough, the model may learn patterns

that are specific to the training data and not generalizable to new data.

The PROSER[1] method addresses the problem of overfitting by using a technique called manifold mixup. Manifold mixup[3] is a technique that mixes two images from different classes together to create a new image that is from a novel class. This allows the model to learn to distinguish between known and novel classes, and it can help to prevent the model from overfitting the training data.

The PROSER[1] method uses a combination of both calibration and open-set probability to improve the performance of closed-set classifiers on open-set tasks. The method uses manifold mixup[3] to generate novel class data, and then uses classifier placeholders to learn the invariant information between target and non-target classes. This allows the model to learn to distinguish between known and novel classes, and it can help to prevent the model from overfitting the training data.

The PROSER[1] method has been shown to be effective in addressing the problem of overconfident predictions in deep learning models. The method has been shown to improve the performance of deep learning models on a variety of open-set recognition datasets.

2. Methodology

Closed-Set Classification:

$D_{tr}^{c} = \{(X_i, Y_i)\}_{i=1}^{m}$ and $D_{te}^{c} = \{(X_i, Y_i)\}_{i=1}^{n}$ are the training and test sets for closed set classification, where $X_i \in \mathbb{R}^D$ is the training instances and $Y_i \in \{1, 2, 3..., K\}$ are the corresponding class labels.
Here we have to fit a model $f(x) : X \rightarrow Y$ that minimizes the expected risk, which is the probability that the classifier misclassifies an instance.
Mathematically, we need to find :

$$\mathbf{argmin}_{f \in H} \mathbf{Ex}(\mathbf{I}(\mathbf{y} \neq \mathbf{f}(\mathbf{x}))$$

where H is the hypothesis space, $\mathbb{I}$ is the indicator function. Also assume that the model f is composed of embedding function $\psi : \mathbb{R}^D \rightarrow \mathbb{R}^d$ and weight matrix $W = [w_1, w_2, ..., w_K]$

Open-set Recognition:

For open set recognition the model is still trained on D_{tr}^{c}, but the test set will contain novel classes, so modified test set is $\tilde{D}_{te}^{c}$ and $\tilde{Y} = \{1, 2, 3, ..., K, K+1\}$ where $K+1$ is the label for the novel classes.

Data Placeholders:

Data placeholders are used to anticipate novel classes. This is done by mixing known class data with novel class data. The novel class data is generated using a technique called manifold mixup. Manifold mixup is a technique that mixes two images from different classes together to create a new image that is from a novel class. This allows the model to learn to distinguish between known and novel classes.Two main characteristics of data placeholders are:
1)This means that the novel class data should be distributed in a way that is different from the distribution of the known class data. This will help the model to

learn to distinguish between known and novel classes.
2)The generating process should be quick. This is important because the model needs to be able to learn to distinguish between known and novel classes quickly. If the generating process is too slow, the model may not be able to learn to distinguish between known and novel classes effectively.
we choose two images from different class and mix them up at the middle layer:

$$\tilde{X}_{previous} = \alpha\psi_{previous}(X_i) + (1-\alpha)\psi_{previous}(X_j); y_i \neq y_j$$

where $\alpha \in [0, 1]$ is sampled fromk Beta distribution.
The loss function is used to train the model to distinguish between known and novel classes.

$$l_2 = \sum_{(X_i, X_j) \in D_{tr}^c} \mathrm{L}([\mathrm{W},\ \tilde{w}]^t \psi_{post}(\tilde{X}_{previous}), K+1)$$

$$\tilde{X}_{previous} = \alpha\psi_{previous}(X_i) + (1-\alpha)\psi_{previous}(X_j); y_i \neq y_j$$

The loss function is based on the idea that the embedding of an image from a novel class will be different from the embedding of an image from a known class. The loss function is designed to minimize the distance between the embeddings of images from known classes and the embeddings of images from novel classes.
Manifold mixup[3] is able to leverage interpolations of deeper hidden representations. Deeper hidden representations are more abstract representations of the data, which means that they can capture more complex patterns. This allows manifold mixup to generate novel patterns that are more representative of the novel distribution

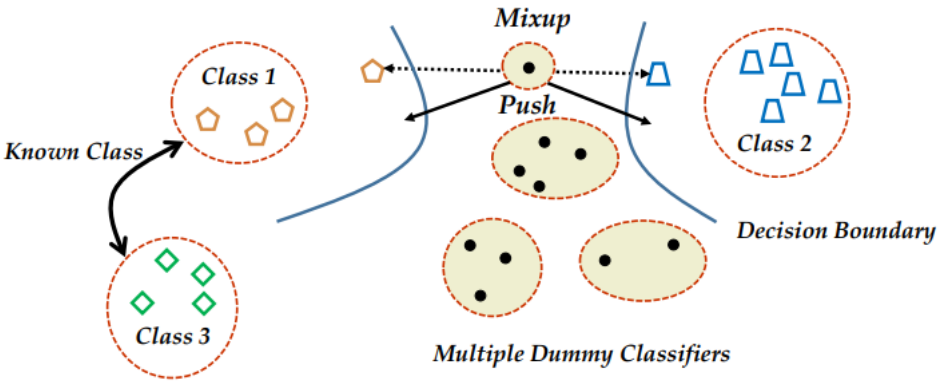

Figure 2: learning data placeholders

The visualization of data placeholders in Figure 2 shows how the mixed instances push the decision boundary in the embedding space towards the composition class y_i and y_j. This means that the model is forced to learn to distinguish between the two classes more effectively.

```
[ ]  class CustomMNISTDataset(torch.utils.data.Dataset):
        def __init__(self, root, transform = None, train=True):
            self.mnist_data = datasets.MNIST(root=root, train=train, transform=transform, download=True)
            self.train = train
            self.resize_transform = transforms.Resize((224, 224))

            if train:
                self.filtered_indices = [idx for idx, label in enumerate(self.mnist_data.targets) if label < 6]
            else:
                self.filtered_indices = [6 if label >= 6 else label for _, label in self.mnist_data]
            #self.transform = transform
            #self.to_pil = ToPILImage()
            #self.grayscale = Grayscale(num_output_channels=3)

        def __len__(self):
            return len(self.filtered_indices)

        def __getitem__(self, idx):
            if self.train:
                filtered_idx = self.filtered_indices[idx]
                image, label = self.mnist_data[filtered_idx]
            else:
                image, _ = self.mnist_data[idx]
                label = self.filtered_indices[idx]



            return image, label
```

Figure 3: Making Dataset

```
[2]  from torch.utils.data import DataLoader

    # Define the transformations
    transform = transforms.Compose([transforms.ToTensor()])

    ## train_test dataset
    train_dataset = CustomMNISTDataset(root='path_to_mnist_dataset12', transform=transform, train=True)
    test_dataset = CustomMNISTDataset(root='path_to_mnist_dataset12', transform=transform, train=False)


    ## Train_Testest Dataloader
    train_data_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

    test_data_loader = DataLoader(test_dataset, batch_size=32, shuffle=True)
```

Figure 4: Dataloader

```
[17] import matplotlib.pyplot as plt
     # Fetch a batch of images and labels from the data loader
     images, labels = next(iter(test_data_loader))

     # Display the images and labels
     fig, axs = plt.subplots(1, 10, figsize=(10, 5))
     fig.tight_layout()

     for i in range(10):
         img = images[i].squeeze()
         lbl = labels[i].item()
         axs[i].imshow(img, cmap='gray')
         axs[i].set_title(f"Label: {lbl}")
         axs[i].axis('off')

     plt.show()
```

Figure 5: Plotting Test Dataset

3. Result:

Model	Dataset	Accuracy
Resnet18	MNIST	60.12%
Data Placeholder	MNIST	76.80%

4. Conclusion:

Thia method is effective in addressing the two main challenges of open-set recognition. The method has been shown to improve the performance of deep learning models on MNIST dataset.
a number of interesting future works that could be done. One interesting future work would be to extend the PROSER method to stream data scenarios. This would allow the PROSER method to be used in applications where the data is coming in real time.

5. References:

[1] Zhou, Da-Wei, Han-Jia Ye, and De-Chuan Zhan. "Learning placeholders for open-set recognition." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2021.
[2] Bendale, Abhijit, and Terrance Boult. "Towards open world recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.
[3] Verma, Vikas, et al. "Manifold mixup: Better representations by interpolating hidden states." International conference on machine learning. PMLR, 2019.
[4] Bendale, Abhijit, and Terrance E. Boult. "Towards open set deep networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
[5]