# Study of Out-of-Distribution Detection for Classification Based on Deep Learning

# 博 士 学 位 論 文

論文題目　Study of Out-of-Distribution Detection for Classification Based on Deep Learning

(深層学習を用いたクラス分類における分布外入力の検出に関する研究)

提　出　者　東北大学大学院情報科学研究科

システム情報科学　専　攻

学籍番号　B7ID2502

氏　名　Techapanurak Engkarat

| | | |
|---|---|---|
| 指　導　教　員<br>Advising Professor<br>at Tohoku Univ. | Professor Okatani Takayuki | [✓ 教　授]<br>准教授 |
| 学位論文指導教員<br>Dissertation Advisor | | 教　授　講　師<br>准教授　助　教 |
| 審　査　委　員<br>（○印は主査）<br>Dissertation Committee<br>Members Name marked with<br>"○" is the Chief Examiner | ○ Prof. Okatani Takayuki<br>1 Prof. Hashimoto Koichi<br>3 Assoc. Prof. Kagami Shingo<br>5 | <br>2 Prof. Takizawa Hiroyuki<br>4<br>6 |

TOHOKU UNIVERSITY
Graduate School of Information Sciences

# Study of Out-of-Distribution Detection for Classification Based on Deep Learning
# (深層学習を用いたクラス分類における分布外入力の検出に関する研究)

A dissertation submitted to the department of
System Information Science in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy
in
Information Sciences

by

TECHAPANURAK Engkarat (ID No.: B7ID2502)

January 15, 2021

# Study of Out-of-Distribution Detection for Classification Based on Deep Learning

TECHAPANURAK Engkarat (ID No.: B7ID2502)

## Abstract

Deep neural network (DNN) achieves human-level performance in many applications and gains attention from industries. However, the question remains at how to integrate DNNs into the real world safely. It is reported that the performance of the DNNs unexpectedly drops in some situations, e.g., when the environments radically change from the training time. Decisions made concerning the predictions of DNN possibly cause damage. This problem is crucial in the task that relates the human life. Autonomous driving vehicles are one of the examples. To guarantee passenger safety, it must be noticeable before DNNs run into failure. Thus, the decision towards the problem can be handled by the human expert.

One source of failure in the open environment is when the model confronts the input image that consists of an unfamiliar feature far from its training dataset. It is called *out-of-distribution (OOD)*. It is reported in many studies that the prediction quality deteriorates regarding the OOD. Indeed, the OOD samples can be roughly categorized into two conditions. First, the samples belong to the unknown class that is not included in the training dataset. Second, the samples are of the known classes whose feature differs from the seen samples in the training phase. The model should neglect the former since they cause the prediction to be wrong; relying on these predictions might cause damage by the consequent action. The latter is more problematic because the samples belong to one of the known classes. The prediction can be partially (in)correct. To handle them, the model needs to know how the expected accuracy deteriorates given that input images. Thus, it is possible to reject the prediction that has expected accuracy lower than the threshold or notify the user that the prediction is likely to be wrong.

In the first part of this work, we tackle the OOD detection in the first condition, i.e., the sample belongs to the unknown class. The baseline method has been evaluated using the probability of the selected class. It works fairly well to detect the OOD samples. Consequent methods are proposed to improve the detection performance so far. Some of the high performing methods assume that they can access a small part of the OOD dataset and involve it in the hyperparameter tuning. Recently, many studies point out that this assumption does not apply in general since the OOD samples may come from a different distribution and cause the detection performance to be unexpectedly low. We proposed the OOD detection method that is free from the hyperparameter tuning. The proposed method replaces the dot product with the cosine similarity and the scale prediction module. We use the cosine similarity as the measurement to detect the OOD samples. We found that the OOD detection performance is significantly improved with sacrificing a slight classification accuracy.

The second part re-evaluates the well-known methods in a more practical situation and re-organizes the OOD detection cases. Although many OOD detection methods have been proposed so far, most of them evaluate their performance on the experimental datasets, e.g., CIFAR-10/100 and Tiny ImageNet. The effectiveness of the methods on the real-world dataset is unknown; they differ in many aspects, e.g., image size and object saliency. We also separate the evaluation into three different cases, i.e., irrelevant input detection, novel class detection, and domain shift detection. Irrelevant inputs and the novel classes belong to the first condition where these samples are from the unknown class; however, their difficulty is different, causing the evaluated methods to perform differently in two cases. The domain shift detection is in the second condition. We apply the OOD detection method and use its measurement to predict the deterioration of the classification accuracy. Our result confirms two conclusions. First, using the network with cosine similarity performs consistently high performance in most cases. Second, fine-tuning the network that is pre-trained from the ImageNet dataset significantly contributes to the improvement.

Lastly, in the third part, we address the concrete relationship to bridge the gap between in-distribution (ID) and the OOD. It has been proved that utilizing a large source of the outlier as the assumable OOD examples can improve the OOD detection performance. This can be considered the auxiliary dataset representing the high entropy prediction in the categorical distribution, i.e., the uniform random. We propose to replace the auxiliary dataset with the synthetic corrupted dataset. We first train the model in the standard manner and observe the empirical accuracy of the corrupted image datasets, using which we create a soft target label from them. We then train the model again from scratch using the original ID samples and the corrupted samples with the soft labels. The experimental result shows that our method outperforms the state-of-the-art method while maintaining a classification accuracy similar to the standard network.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

This chapter gives an introduction about the problem we will discuss throughout this dissertation. We introduce an overview to illustrate the territory of the problem in Sec. 1.1. To encourage more understanding, we include the preliminary knowledge related to our study in Sec. 1.2. Lastly in this chapter, the outline of each individual chapter are described in Sec. 1.3.

## 1.1 Overview of Out-of-Distribution Problem

Artificial intelligence (AI) system becomes an important part that backed many advanced technologies nowadays. It plays an important role in many real-world tasks both in daily life and industries. Although the current AI agents are recognized as *narrow AI*, i.e., they are made to accomplish the narrowly defined task, their performance reported outperform the human in many tasks, such as medical diagnosis [8] and speech recognition [9].

However, there remain concerns in deployment to the real world. One of them is a safety concern. It is reported in many studies that deep neural network (DNN) may silently run into a failure in some situations [10]. Suppose we launch the autonomous car into the real-world environment and it wrongly detects the pedestrians as the street. This possibly causes damage to the system and human life. Thus, studying about realizing the fail state of DNN is a preventive strategy. Once it is detected, we could handle the situation effectively before it causes damage.

Researchers study the failure of DNN in many aspects. In general, they are searching for a way to allow the DNN to express confidence of the prediction accurately, such that the users can effectively manage their risk. For example, suppose we consult the AI stock investment advisor to decide on selling or buying stock. The investor can decide not to follow or follow with just a small investment once the system advises with low confidence. It is considered "calibrated" when the model gives the confidence that expresses its correctness in predictions [11, 12]. Bayesian model is also deployed to get the uncertainty estimation. It can produce uncertainty from the randomness of its parameters. Indeed, modern DNN usually uses the Bayesian approximation, e.g., MC dropout [13], due to the difficulty of training the deep Bayesian model. It shows an effective performance in the real-world applications [14–16].

One source of a wrong prediction is *out-of-distribution (OOD).* In the ideal situation, model trained on training samples, $x_{tr}, y_{tr} \sim \mathcal{D}$, is expected to be the expert in classifying test samples from the same distribution, $x_{te}, y_{te} \sim \mathcal{D}$. However, it is also possible that the test samples given in the real-world environment are not from the same distribution, $x_{te}, y_{te} \sim \mathcal{D}^*$, where $\mathcal{D} \neq \mathcal{D}^*$. The samples from different distribution are considered OOD for the model. Eventually, it leads to compromised performance in the applications. Thus, we study the OOD problem in the classification model, such that we can actively handle the OOD samples.

Suppose we launch a mobile application that aims to help the user classify the food from the image and inform about the nutrition. However, the image of the irrelevant object might be accidentally given to our application, e.g., an image of the human or the animal. In this case, making a prediction on these images might reduce the trust in our application. In this situation, detecting it as an irrelevant object is considered a better user experience.

The OOD problem can cause even more crucial problems in the real world. As we have mentioned in the example case of the autonomous car, the OOD sample is a threat in its deployment too. In the training time, we definitely have a limited set of samples for training. The model probably learns to recognize the objects on the street from the feature of each object and becomes an expert in this task. In the inference time, the image coming to the model is unlimited due to the variety in the real-world object. Further, the image quality might decline at some point in time such that the classification power reduces. Suppose there is a sofa lying on the street. Since the model has learned the objects that are generally located on the street, the possibility to see any sofa in the training time is low or zero. It is important for the model to realize the sofa in an unknown object such that it does not make a wrong prediction to it, e.g., predict it as a road and drive on it.

To the OOD samples, it might be acceptable if the model gives an accurate uncertainty estimation such that OOD samples should obtain high uncertainty in predictions. Unfortunately, it is not the case. It is known that the natural uncertainty measurements of the classification task, i.e., maximum softmax probability and predictive entropy, and the Bayesian approximation methods produce a fair performance in the OOD detection [2,17,18]. Consequent methods have been proposed to improve the OOD detection performance so far. Some of them are especially designed for the classification model [3, 4, 19, 20], while the other can be used regardless to any tasks [21–24]. Note that the classification based OOD detection methods tend to achieve better detection performance and are easier to generalize to larger image size.

**Types of OOD sample** Suppose the classification model learns to predict the object class of training samples, $\{(x_i, y_i)\}_{i=1}^N$, where $(x_i, y_i)$ is the image and label pair of $i$-th sample. The label $y_i \in Y$, where $Y$ contains $K$ known classes, indicates the category that sample $x_i$ belongs to. It is possible to categorize OOD samples into two categories.

1. **OOD sample that does not belong to the known classes** $x_i, y_i \sim \mathcal{D}^*; y_i \notin Y$ - Most of the OOD detection studies focus on this type of the OOD sample. It is when the OOD is a sample of the class that is not included in the model training, i.e., unknown class. To the human perception, it is easy to realize the OOD samples in this type but it is unexpectedly difficult for the modern DNN. However, predicting a sample $x_i$ as one of the known classes will definitely lead to a wrong prediction. Thus, we can simply reject all the predictions for this type of sample to prevent the mistake.

2. **OOD sample that belongs to the known classes** $x_i, y_i \sim \mathcal{D}^*; y_i \in Y$ - This type of OOD sample is rarely considered in the OOD detection field of

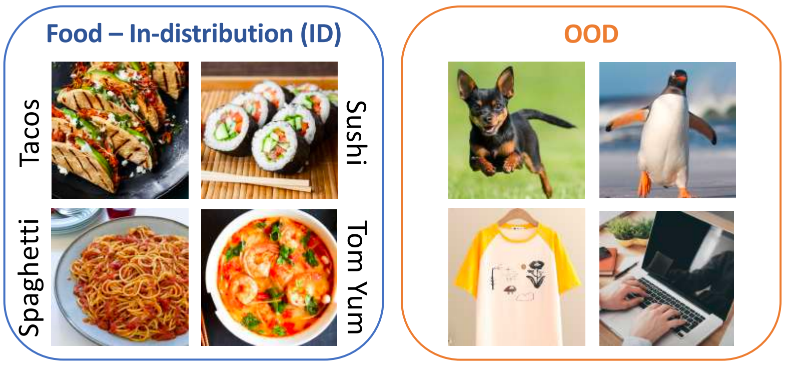


Figure 1.1: Examples of OOD that do not belong to known classes. (Image source: `https://cutt.ly/6j6PsKq`)

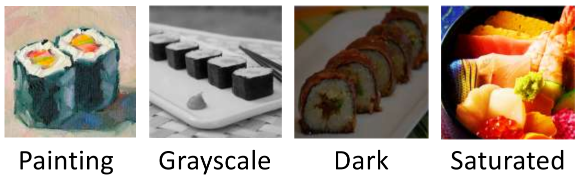


Figure 1.2: Examples of OOD that belong to known classes. (Image source: `https://cutt.ly/fj6PJWI`)

study. It is when the OOD sample is drawn from a different distribution, but it is still a member of one of the known classes. For example, the model learns to classify $K$ object categories of the images taken by the DSLR camera, but the test images are obtained from the low-resolution camera and are corrupted with the noise. The difference in the image domain deteriorates the performance in classification. Thus, it is important to predict the deterioration such that the practitioner can realize and manage the risk.

**Hyperparameter Tuning in OOD Detection Task** Some OOD detection methods are proposed with a requirement to access a small amount of the OOD samples. These samples are used to define the hyperparameter(s) such that the methods can effectively detect the OOD samples. However, this logic does not apply in general. The study by Shafaei et al. [1] raises the issue to this assumption. The variety of the OOD samples hinders this assumption from being effective in practice because the OOD samples can be very different from each other. Suppose the model decides its hyperparameter using the OOD validation samples, i.e., $\{(x_i, y_i)|x_i, y_i \sim \mathcal{D}_v^*\}$, where

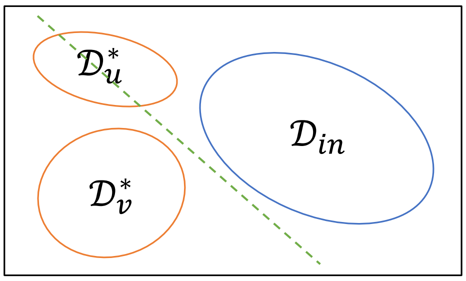


Figure 1.3: The parameter decided by using validation OOD samples does not guarantee its generalization to the unseen OOD samples. $\mathcal{D}_{in}$ represents the in-distribution dataset and $\mathcal{D}_v^*$ and $\mathcal{D}_u^*$ are the validation OOD and the unseen OOD datasets, respectively. The green dashed line illustrates the decided hyperparameter as a dependence of the validation OOD dataset. It does not separate in-distribution from the unseen OOD effectively.

$\mathcal{D}_v^*$ is the validation OOD distribution, it does not guarantee that the decided hyperparameter will be appropriate for the unseen OOD, i.e., $\{(x_i, y_i)|x_i, y_i \sim \mathcal{D}_u^*\}$, where $\mathcal{D}_u^*$ is the unseen OOD distribution and $\mathcal{D}_v^* \neq \mathcal{D}_u^*$, as shown in Fig. 1.3. Most of the recent methods are proposed with avoiding the necessity of the validation OOD samples [5, 25–27].

## 1.2 Preliminaries

To encourage more understanding, we introduce the preliminary knowledge that is relevant to our work in this section.

### 1.2.1 Maximum a Posteriori

The statistical model predicts the output, $y$, from the input, $x$, using its learnable parameter, $\theta$. The proper solution for the learnable parameter is found by

$$\hat{\theta} = \arg\max_{\theta} \mathrm{P}(\theta|\mathcal{D}_{tr}),$$

where $\theta$ is the model's parameter, $\mathcal{D}_{tr}$ is the training dataset, and $P(\theta|\mathcal{D}_{tr})$ is a posterior term of the Bayes' rule. It can be interpreted as the parameter that maximizes the likelihood given the training dataset. However, calculating $P(\theta|\mathcal{D}_{tr})$ is intractable.

According to the Bayes' rule, we know that

$$\mathrm{P}(\theta|\mathcal{D}_{tr}) = \frac{\mathrm{P}(\mathcal{D}_{tr}|\theta)\mathrm{P}(\theta)}{\mathrm{P}(\mathcal{D}_{tr})}.$$

To come up with the tractable solution, we convert the problem to

$$\hat{\theta} = \arg\max_{\theta} \frac{\mathrm{P}(\mathcal{D}_{tr}|\theta)\mathrm{P}(\theta)}{\mathrm{P}(\mathcal{D}_{tr})}.$$

Since the solution of the $\arg\max_{\theta}$ is independence of the denominator term, we can then discard it. Thus, the formulation is given as

$$\hat{\theta} = \arg\max_{\theta} \mathrm{P}(\mathcal{D}_{tr}|\theta)\mathrm{P}(\theta),$$

where the prior probability term, $P(\theta)$, is usually represented by the weight regularization. Finally, we use the parameter found by the optimization in the prediction, $y \sim P(y|x, \theta)$.

### 1.2.2 Negative Log Likelihood

The general concept of a predictive model is to allow the model to make a prediction towards an incoming sample $x$. The output (i.e., prediction) of the model will be a parameter(s) of the pre-defined probability distribution. For the *regression model*, the model that outputs a concrete number as the prediction, the pre-defined probability distribution is the Gaussian distribution. For the *classification model*, the model that predicts the category of the input sample, it is categorical distribution.

*Gaussian distribution* is the probability distribution that depends on two parameters, i.e., mean ($\mu$) and standard deviation ($\sigma$). The mean indicates the center of where the values are distributed. The standard deviation represents how the value is likely to deviate from the center. The probability density function (PDF) is given as

$$\mathrm{P}(x) = \mathrm{Norm}_x[\mu, \sigma^2] = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\,[\frac{-(x-\mu)}{2\sigma^2}].$$

*Categorical distribution* is the probability distribution defined for a $K$ possible outcomes, e.g., rolling dice. It requires $K$ parameters where each of them represents the probability of the individual drawn sample belongs to each category, $\mathbf{C} = [c_1, c_2, \ldots, c_K]$ and $\sum_i^K c_i = 1$. The PDF is defined as

$$\mathrm{P}(x = k) = \mathrm{Cat}_x[\mathbf{C}] = \prod_i^K c_i^{e_i},$$

where $e_i \in \{e_1, e_2, \ldots, e_K\}$ where $e_k$ is one and all the others, $\{e_i | i \neq k\}$, are zero.

Training the machine learning model is the optimization process performed concerning the objective function. Specifically, the model learns such that the objective function is satisfied, i.e., minimized, maximized, or getting to the equilibrium, by its parameter $\theta$. In general, the objective function is *likelihood function*, $\prod_i^N P(x_i; \theta)$, where $x_i$ is the $i$-th observed sample, $P(\cdot)$ is the pre-defined PDF, and $\theta$ is the parameter of the PDF. It is the measurement of the goodness of the parameter in the PDF regards to the observed samples. In this sense, the parameter will satisfy our expectation if *likelihood is maximized.* However, it encounters a numerical problem in the calculation since the likelihood tends to approach a low value close to zero when the amount of the observed samples is expanded. The log function is introduced to solve this problem and the formulation becomes *log-likelihood*, given as

$$\mathrm{Log-likelihood} = \sum_i^N \log \mathrm{P}(x_i; \theta).$$

Most of the famous DNN frameworks, e.g., Pytorch or Tensoflow, learning step is performed in the minimization manner. Thus, the objective function eventually becomes negative log-likelihood,

$$\mathrm{NLL} = -\sum_i^N \log \mathrm{P}(x_i; \theta).$$

The NLL of the Gaussian distribution is defined as

$$\mathrm{NLL}_{gaus} = -\sum_i^N [\log \sqrt{2\pi} + \log \sigma + \frac{(x_i - \mu)^2}{2\sigma^2}],$$

and the NLL of the categorical distribution is defined as

$$\text{NLL}_{cat} = -\sum_{i}^{N}\sum_{i}^{K} e_i log c_i.$$

### 1.2.3 Uncertainty Estimation

As we mentioned in the previous section, in the real-world application, it is crucial to allow the model to realize when its prediction is likely to be a wrong prediction. This is an important mechanism to help the practitioner decides whether to trust the prediction from the predictive model. This concept is an *uncertainty estimation.*

The uncertainty can be categorized by its cause. Two types of uncertainty are mentioned in many studies, i.e., Aleatoric and Epistemic uncertainty [14]. The aleatoric uncertainty is the natural uncertainty that is caused by the observed data. It can also be the noise of how to collect or measure the data. For example, suppose we collect the data by measuring the length of the box with the ruler. The length will be observed by human eyes. Observing the length multiple times by the same or different persons may also cause different observed values; it is varied in a limited range.

Aleatoric uncertainty can be estimated by the natural property of the posterior distribution, i.e. the PDF. In the regression problem, the aleatoric uncertainty is given by the predicted variance [14] with regard to the NLL optimization; the definition of the variance is that it conveys how the data diverge from its mean. Thus, high predicted variance indicates high uncertainty. In the classification, the aleatoric uncertainty is represented by the variational ratio or the entropy [28].

Epistemic uncertainty expresses the other type of uncertainty that is caused by the model parameter. This type of uncertainty accounts for the lack of training data. Thus, it is reduced once the dataset is increased. The Bayesian model, or its approximation, e.g., dropout, is utilized to estimate the epistemic uncertainty.

### 1.2.4 Anomaly Detection

Anomaly detection is an approach to detect the sample that differs or diverge from the normality, called *anomaly.* In general, the anomaly indicates some improper behaviors or events, e.g., the attack on the network system. It is simple to model the statistic from the data to see the anomalous sample from the low dimensional data. The problem becomes more challenging when the data is in a high dimension. Note that anomaly is also referred to as the other terminology, e.g., outlier and novelty.

The basic idea to detect the anomaly is to model the $P(x)$ that output the likelihood of a sample. For the high-dimensional data, we can utilize the generative model as the tool to model $P(x)$. A simple way to perform the generative model is to use the autoencoder model. The anomalous sample can be determined from the intermediate layer, i.e., bottleneck layer, or from the reconstruction loss. The other types of neural network are also proposed, e.g., PixelCNN [29], GAN [21], and Glow [22].

Image anomaly detection is widely applied to many real-world problems. It is also used to determine the defection in the manufacturing goods [30]. The method performs detection by observing the image of the product. The detected product is considered an anomaly in this case.

### 1.2.5 Deep Neural Network

Artificial Neural Network (ANN) was introduced by Frank Rosenblatt in 1958 [31], inspired by the neuron in the human brain, called *perceptron*. It was made to mimic how humans make a decision or learn to recognize the object. It resembles the mechanism of how the brain works, i.e., dendrites and axons. Dendrites are responsible for delivering the information into the neuron. Axons export the output and let it flows across the synapse. Multiple neurons interact with each other and exchange the information in one decision making. The ANN also models the decision making by having the input information flows in, and delivers it out after processing. It was used to classify the image of human into *man* or *woman* classes. It did not work well at that time.

In 1986, David et al. [32] proposed backward propagation as a learning method for the multi-layer perceptron. It was significant progress in the research of artificial neural networks. It allows the ANN to learn effectively from its error. This became a core idea in any neural network nowadays.

A modern neural network is built more complicated. Both hardware and software technologies in this era encourage the advance in neural network research. GPU becomes necessary hardware to execute the model. The capacity that we have in the modern computer is enough for the design of the deeper neural network these days. We know it as *deep neural network (DNN)* or *deep learning*. It shows excellent performance and even outperforms humans in many tasks. It can be applied in a wide range of problems such as supervised learning, unsupervised learning, reinforcement learning, generative adversarial network (GAN), etc.

A significant milestone was made by DNN when it won the ImageNet Large Scale Visual Recognition Competition (ILSVRC) in 2012. Krizhevsky et al. tackled the image classification problem with the DNN model called AlexNet and radically changed the direction of the research community [33]. Specifically, the special type of neural network used in the AlexNet is *Convolutional Neural Network (CNN)*. It utilizes the DNN module that calculates the output in a spatial invariant manner. Before that, it was difficult to achieve high accuracy in this task by the traditional computer vision approach. The performance made by the AlexNet at that time produced a big gap in the improvement from the others.

The world got excited again by the rising of Deepmind's AlphaGO, an AI that plays Go better than professional human, at the beginning of 2016. Go is an ancient board game that is originally played in China from 2,500 years ago [1]. Two players have to compete with each other by occupying the territory and avoid getting captured by the opponent. The score is eventually counted at the end of the game by the occupied area subtracted by the number of the captured stones. It is known that this game is difficult for any computer program to competitively play with the professional human player by the traditional AI algorithm such as Minimax [34]. AlphaGo combines two mechanisms in their calculation, i.e., policy network and value network. Former generates a set of multiple moves that possibly gain the score using reinforcement learning. Latter estimates the goodness of that move considering the overall situation using the regression model.

Many parts of neural networks are introduced and become necessary to push the advance in this technology. In what follows, We describe the underlying idea of how they work and why it is crucial for modern neural networks.

[1]`https://en.wikipedia.org/wiki/Go_(game)`

### Fully Connected (FC) Layer

Fully connected (FC) layer is a linear approximation module that takes all elements in the input into account. This module obtains the input and applies the linear equation by its learnable parameters, i.e., weight and bias, to approximate the output. The input and output can be a vector of any dimension. Suppose the input $\mathbf{x} = \{x_1, x_2, \ldots, x_m\}$ is the vector of $m$ dimensions and the output $\mathbf{y}$ contains $n$ dimensions, the calculation is

$$\mathbf{y} = \mathbf{W}^{\mathrm{T}}\mathbf{x} + \mathbf{b} = \begin{bmatrix} w_{1,1}x_1 + w_{2,1}x_2 + \cdots + w_{m,1}x_m + b_1 \\ w_{1,2}x_1 + w_{2,2}x_2 + \cdots + w_{m,2}x_m + b_2 \\ \vdots \\ w_{1,n}x_1 + w_{2,n}x_2 + \cdots + w_{m,n}x_m + b_n \end{bmatrix},$$

where $\mathbf{x}$ is the input, $\mathbf{W} \in \mathbb{R}^{m \times n}$ is the weight parameter, and $\mathbf{b} \in \mathbb{R}^n$ is the bias parameter. The special case of the FC layer when $n = 1$ is the linear regression model.

Stacking multiple FC layers increases the ability to approximate the value. However, naively stack the layers has an issue in mathematical aspect. It is necessary to apply the activation function between each layer. The detail will be described in the following section.

### Activation Function

The key characteristic of the DNNs is that they are comprised of the multiple layers such that it promotes the predictive power. Indeed, naïvely stacking the FC layers does not improve much result in model learning because of the mathematical issue. Specifically, the two-layer neural network calculation is

$$\begin{aligned} y &= \mathbf{W}_2^{\mathrm{T}}(\mathbf{W}_1^{\mathrm{T}}\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \\ &= \mathbf{W}_2^{\mathrm{T}}\mathbf{W}_1^{\mathrm{T}}\mathbf{x} + \mathbf{W}_2^{\mathrm{T}}\mathbf{b}_1 + \mathbf{b}_2 \\ &= \tilde{\mathbf{W}}^{\mathrm{T}}\mathbf{x} + \tilde{\mathbf{b}}, \end{aligned}$$

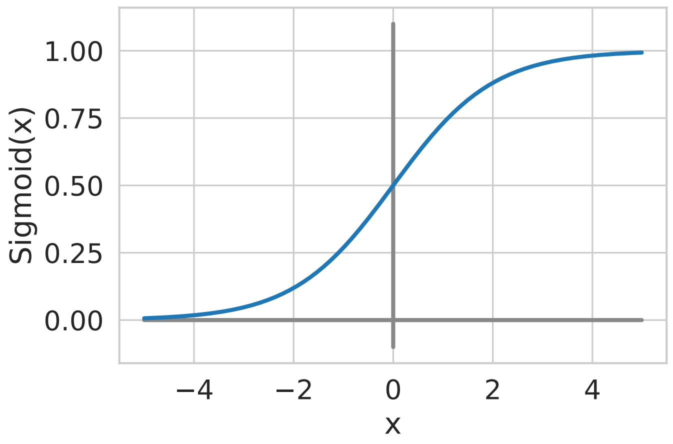


Figure 1.4: The plot of sigmoid function.

where $\tilde{\mathbf{W}}^{\mathrm{T}}$ is the linear combination of the weight, i.e., $\mathbf{W}_2^{\mathrm{T}}\mathbf{W}_1^{\mathrm{T}}$, and $\tilde{\mathbf{b}}$ is the adjusted value of the bias, i.e., $\mathbf{W}_2^{\mathrm{T}}\mathbf{b}_1 + \mathbf{b}_2$. Thus, in this case, having two layers does not increase the ability to approximate the output.

The activation function is introduced to solve this problem. Being the non-linear function, it locates between layers to break their linearity. By the insertion, the two-layer neural network becomes

$$y = \mathbf{W}_2^{\mathrm{T}}\sigma(\mathbf{W}_1^{\mathrm{T}}\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2,$$

where $\sigma(\cdot)$ represents the activation function.

The well-known activation functions, that are mostly applied, are given as follows.

**Sigmoid Function or Logistic Function** Sigmoid function is the well-known activation function that is also used in the logistic regression model. It converts the input of real value, $(-\infty, \infty)$, to the probability-like output, $[0, 1]$, as shown in Fig. 1.4. The formulation is given as

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}.$$

**Hyperbolic Tangent Function** Hyperbolic tangent function (tanh) is one of the hyperbolic functions, i.e., trigonometric functions defined for the hyperbola. It con-

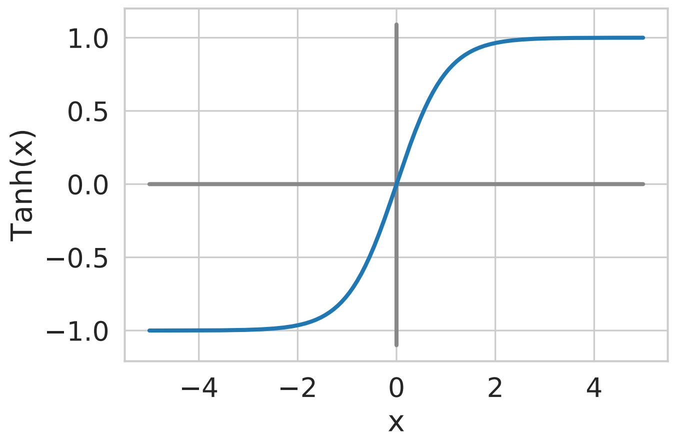


Figure 1.5: The plot of hyperbolic tangent (tanh) function.

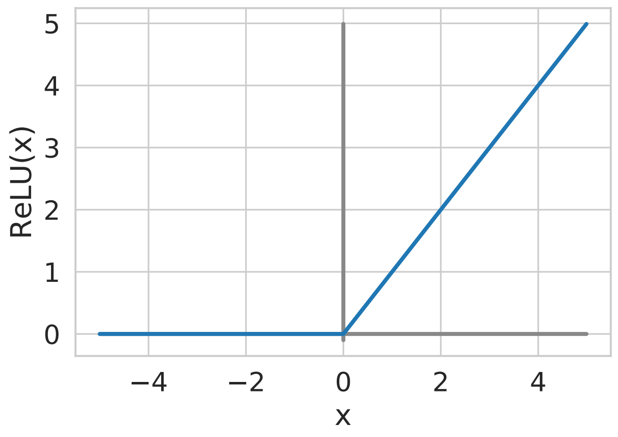


Figure 1.6: The plot of rectified linear unit (ReLU) function.

verts the input of real value, $(-\infty, \infty)$, to the output of range $[-1, 1]$. This plot between input and output is shown in Fig. 1.5. The function is defined as

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

**Rectified Linear Unit (ReLU)** Although the above two activation functions can introduce non-linearity to the model, there remains a problem. In general, the optimization is performed utilizing the gradient (i.e., the slope) of the function. The function that produces the output with the slope equals or close to zero will hinder optimization. The problem is called *gradient vanishing*. Thus, for the sigmoid and hyperbolic tangent function, the model encounters this problem when the input value is far from zero.

The ReLU function can solve this problem. ReLU is the non-linearity function that outputs the value identical to its input when the input value is higher than zero; otherwise, it outputs zero. Concretely, the function is

$$\text{ReLU}(x) = max(0, x).$$

The gradient that is caused by this function is 1 if the input is more than zero, and it is 0 when the input is less than zero.

### Convolutional Neural Network (CNN)

The traditional multi-layer neural network is comprised of the FC layers. It learns to recognize the patterns from the data by the whole information at a time. Although it works well for some types of data, e.g., tabular data, it does not work well on image data.

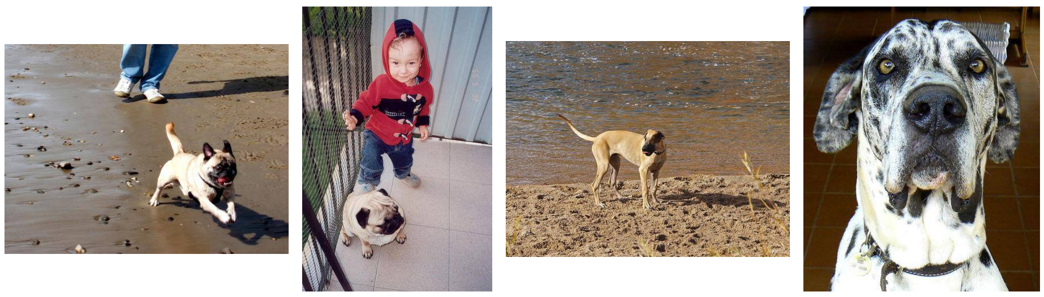


Figure 1.7: Examples of dog images. The main interested areas are located differently in different images. They differ in spatial dimension, size of object, surrounding environments, etc. (Image source: `https://www.kaggle.com/c/dog-breed-identification`)

Images contain a more complicated pattern than the tabular data. Since the image has spatial dimensions, i.e., height and width, the informative content can be varied in the image. For instance, the image sample that belongs to class *Dog* must present the whole or some part of the dog in the image. Comparing two different samples, the dog might be located in a totally different spatial position and context, as shown in Fig. 1.7. Thus, the model must be invariant to changes in the spatial dimension. The

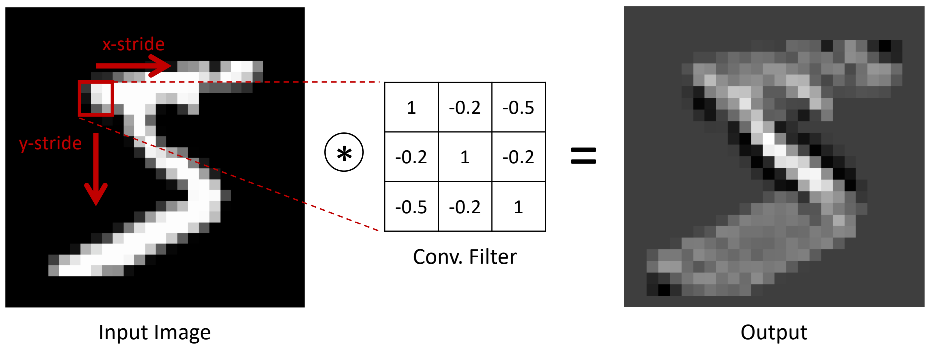


Figure 1.8: Two dimensional convolution filter in CNN. The convolutional filter move throughout the image in x and y axes. It performs the element-wise multiplication in each area.

FC layer does not satisfy this requirement since it takes the whole image to calculate the output.

CNN consists of the module called the 2-dimensional convolution layer. This module is designed to capture the feature of the image such that its output is invariant to the spatial dimension. Specifically, it calculates the output from a small and limited area in the image, e.g., 3×3 or 5×5 pixels, but the calculation is performed throughout the spatial dimension of the image by moving the weight, so-called *filter*, along with height and width, as shown in Fig. 1.8. The step of moving the filter is called "stride".

### Pooling Layer

Similarly to the convolution layer, the pooling layer applies the filter throughout the input. The difference is that it does not calculate the output using the parameter, but it applies the aggregation function to the input data. *Max pooling layer* and *average pooling layer* are the pooling layers that are mostly used in DNN.

The pooling layer is mostly used to reduce the spatial dimension of the input. In this sense, it is usually applied to the input using the stride larger than 1, e.g. $stride = 2$, such that the sliding of the filter will produce the output with lower height and width.

The global pooling layer is a special type of pooling layer. It is attached at the end of the convolution layers to aggregate the entire spatial dimension of the input, and produce the output of $1 \times 1$ pixels. The output is then possible to be forwarded to the FC layer because the spatial dimension has been removed.

The other variation of the pooling layer is the covariance pooling layer [35]. It utilizes the covariance to replace the average function.

### Batch Normalization (BN)

In modern DNN, the training is executed in batch manner. The batch of training samples is randomly selected from the training dataset and is given to the model. This way of training promotes the generalization of the trained model. However, it encounters the problem when the sampled batches' statistics is fluctuated, i.e., mean and standard deviation. It causes a difficulty to the training, e.g., slow convergence. This problem is called *internal covariate shift* [36].

Batch normalization was first introduced by Ioffe and Szegedy [36] to reduce the internal covariate shift. During the training process, it transforms the input data using empirical statistic of the batch with the standard score formulation,

$$z = \frac{x - \mu}{\sigma},$$

where $x$ is the input sample, $\mu$ and $\sigma$ are the empirical mean and standard deviation observed from the batch. The running mean and standard deviation are tracked from the training phase. They are applied instead of the empirical statistics in the inference time.

The standard score is then modified by the parameterized shifting and scaling. Concretely, the formulation is

$$\hat{z} = \gamma z + \beta,$$

where $z$ is the standard score produced by the previous step, $\gamma$ is the learnable scaling factor, and $\beta$ is the learnable shifting offset.

### Softmax and Temperature Scaled Softmax

Training DNN requires the model to output the probability-like value such that the quality of the prediction can be estimated, i.e., loss value. In fact, the output from the model is the real value that varies from $-\infty$ to $\infty$. The softmax function is generally attached at end of the network to transform the network's output into the probability-like value that varies from 0 to 1. Concretely, the formulation is

$$\text{Softmax}(z)_i = \frac{\exp z_i}{\sum_j^C \exp z_j},$$

where $z_i$ is the $i$-th element of the DNN's output, called *logit*, $C$ is the number of the total classes.

Temperature scaling is also introduced in many tasks such as knowledge distillation [37] and calibration [11]. The temperature scaling can be used to adjust the confidence (probability) of the prediction. The formulation is given as

$$\text{Softmax}(z, T)_i = \frac{\exp [z_i/T]}{\sum_j^C \exp [z_j/T]},$$

where $T$ is the temperature. High temperature encourages the lower confidence, while low temperature increases the confidence.

### Loss Function and Gradient Descent Optimization

The model is trained using the optimization method. Meaning that we search for the appropriate model for the training dataset by obtaining the clue from the observed samples, i.e., the training dataset. In the optimization problem perspective, the step of the optimization performs to satisfy the objective which is quantitatively measured, whether it is minimization or maximization. From DNNs perspective, the objective of the optimization is called *loss function*. The optimization aims to minimize the loss regarding the training samples. In general, the loss function is the negative log-likelihood function. See Section 1.2.2 for more detail.

To reduce the loss, the optimization measures the tendency of the loss value concerning model parameters and adjusts the parameters in the direction that minimizes the loss. This optimization is called *gradient descent optimization*. Gradient is the tendency that is used to adjust the model's parameters; it is the partial differentiation with regard to the parameters, i.e., $\nabla_w = \frac{dLoss}{dw}$, where $\nabla_w$ is the gradient w.r.t. the parameter $w$ and $Loss$ indicates the loss value.

Indeed, the stochastic variation of the gradient descent is adopted in DNNs. The batch of the random samples is selected from the training dataset and is given to the model for training rather than using the entire dataset.

## 1.3 Outline of the Dissertation

In this dissertation, we describe and discuss the problems of the out-of-distribution in the classification model. We also include analyses in many aspects. The outline of the content is described in what follows.

In Chapter 2, we raise the unreasonable scheme of the existing OOD detection methods. As we mentioned earlier in Section 1.1, many OOD detection methods consider the problem in an unrealistic scenario. Specifically, they assume the existence of the validation OOD samples and use them in the hyperparameter tuning before evaluating the detection performance. We argue this assumption. The same argument is also raised in the other paper [1]. Our assumption of the problem of the OOD detection is that the OOD detection method should not depend on the validation set for the hyperparameter tuning. Thus, we propose the cosine network, the novel approach to detect the OOD samples. Furthermore, the proposed method is free of the hyperparameter tuning that is done regards to the OOD detection task. To emphasize the unrealistic of using validation OOD samples, we adopt the evaluation method proposed by [1] which utilizes a more realistic assumption. The results prove that requiring the validation samples is not appropriate; it gives low performance when the validation does not represent the unseen OOD samples well enough. The cosine network outperforms the other methods in this evaluation. It also has competitive performance in the conventional evaluation which allows the unrealistic assumption.

In Chapter 3, we re-evaluate the existing OOD detection method in the practical classification tasks. In the research community, many OOD detection methods show high performance in the experimental dataset, e.g., CIFAR-10/100, MNIST, and SVHN. However, there remains a question if these methods perform well to detect the OOD samples in the realistic images, i.e., the image that has higher resolution, and in the practical OOD cases. We only select the OOD detection methods that are either independent of the hyperparameter tuning or having the relaxation to remove the validation samples requirement. The experiments are separated into three cases. First, we test the model on *irrelevant inputs detection*. It is the most common evaluation that is used in most of the OOD detection studies [2–4]. The irrelevant input is the sample that is out of consideration in the application aspect. Second, we consider more difficult case, i.e., *novel class detection*. Suppose we train the model to classify $k$ breeds of dogs. The $(k + 1)$-th breed is considered a novel class to our model. In the application aspect, this type of sample is possibly the candidate of being collected. They can involve in the improvement of the knowledge for the classification model, e.g., re-train the model with the extended class. Third, we focus on domain shift detection. Domain shifting is a problem that causes compromised performance in real-world applications. It is when the incoming sample is far from the seen training samples; it differs in some aspects, e.g., resolution, lighting condition, and image quality. The domain shifting is represented by both real and synthesized images. We use the OOD detection score to predict the accuracy deterioration such that the practitioners can decide whether to accept the deterioration or to perform the domain adaptation. We found that the cosine network works consistently well in all cases compared to the others.

In chapter 4, we propose novel training to bridge the gap between ID and OOD by the synthesized corrupted images. Cosine network has to sacrifice a slight performance in the classification accuracy to perform well in OOD detection. The proposed method in this chapter can achieve higher performance in OOD detection while the ID accuracy is similar to, or even higher than, the standard network. In general, the training dataset contains the samples that absolutely belong to one of the categories, i.e., $p_c = 1$, where c is the labeled class. Learning with the vanilla approach produces a large gap between each learned categories. It causes difficulty for the DNN to assign low confidence to the OOD samples. Hendrycks et al. [7] proposed to alleviate this problem by using the auxiliary dataset which represents the seen outlier. It is expected to well simulate the incoming OOD samples in the inference time. The effectiveness of this assumption is questionable. Inspired by the experiment in Chapter 3, we utilize the corrupted images to fill this gap in the feature space; the corrupted image is considered shifted from the original training dataset. They involve in the training to represent the area of low confident prediction. Specifically, we assign the soft-label as the target label in the training. We found that the proposed method achieves high performance in OOD detection and outperforms the others in conventional OOD detection. Utilizing the corrupted images also pushes the performance in the ID classification. Besides, we also observe that it improves the calibration of the model.

Finally, in Chapter 5, we summarize the content and provide the conclusion of this dissertation. Furthermore, we state the open problem and the possible future work in this chapter.

# Chapter 2

# Hyperparameter-Free Out-of-Distribution Detection Using Cosine Similarity

## 2.1 Introduction

It is widely recognized that deep neural networks tend to show unpredictable behaviors for *out-of-distribution* (OOD) samples, i.e., samples coming from a different distribution from that of the training samples. They often give high confidence (i.e., high softmax value) to OOD samples, not only to *in-distribution* (ID) samples (i.e., test samples from the same distribution as the training samples). Therefore, it has been a major research topic to detect OOD samples in classification performed by deep neural networks; many methods have been proposed so far [2–4, 19, 27, 38–40].

A problem with the existing methods, especially those currently recognized as the state-of-the-art in the community, is that they have hyperparameters specific to OOD detection. They determine these hyperparameters using a certain amount of OOD samples as 'validation' data; that is, these studies assume the availability of (at least a small amount of) OOD samples. This assumption, however, is unlikely to hold true in practice; considering the definition of OOD, it is more natural to assume its distribution to be unknown. Even when the assumption is indeed wrong, it will be fine if OOD detection performance is insensitive to the choice of the hyperparamters, more rigorously, *if the hyperparameters tuned on the assumed OOD samples generalize well to incoming OOD samples we encounter in practice.* However, a recent study [1] indicates that this is not the case, concluding that none of the existing methods is ready to use, especially for the tasks with high-dimensional data space, e.g., image classification.

In this chapter, we propose a novel method that uses cosine similarity for OOD detection, in which class probabilities are modeled using softmax of scaled cosine similarity. It is free of any hyperparameters associated with OOD detection, and thus there is no need to access OOD samples to determine hyperparameters, making the proposed method free from the above issue. We show through experiments that it outperforms the existing methods by a large margin on the recently proposed test [1], which takes the above issue of hyperparameter dependency into account; it also attains at least comparable performance to the state-of-the-art methods on the conventional test, in which the other methods but ours tune hyperparameters using explicit OOD samples.

It should be noted that a concurrent work [5] also shows the effectiveness of softmax of the scaled cosine similarity for OOD detection. Our method is technically mostly the same, but the present chapter shows several different results/conclusions from their paper. The paper [5] shows a conjecture that the scaling factor of the cosine similarity approximates the probability of an input being in-distribution, contributing to improved detection performance. In this chapter, however, we show empirical evidence that this is not the case. It is also noted that, although recent methods for metric learning [41–46] similarly employ scaled cosine similarity as well, they do not guarantee its effectiveness on OOD detection. There are several differences from them in the output layer's design, which contributes to detection accuracy. Concerning this, we provide a detailed ablation study to clarify the method's differences from common metric learning approaches.

## 2.2 Related Works

### 2.2.1 Uncertainty of Prediction

It is known that when applied to classification tasks, deep neural networks often exhibit overconfidence for unseen inputs. Many studies have been conducted to find a solution to this issue. A popular approach is to evaluate uncertainty of a prediction and use it as its reliability measure. There are many studies on this approach, most of which are based on the framework of Bayesian neural networks or its approximation [13, 17, 47, 48]. It is reported that predicted uncertainty is useful for real-world applications [14–16, 49, 50]. However, it is still an open problem to accurately evaluate uncertainty. There are also studies on calibration of confidence scores [11, 51, 52]. Some studies propose to build a meta system overseeing the classifier that can estimate the reliability of its prediction [53, 54].

### 2.2.2 Out-of-distribution (OOD) Detection

**Detection Methods.**

A more direct approach to the above issue is OOD detection. A baseline method that thresholds confidence score, i.e., the maximum softmax output, is evaluated in [2]. This study presents a design of experiments for evaluation of OOD detection methods, which has been employed in the subsequent studies. Since then, many studies have been conducted. It should be noted that these methods have hyperparameters for OOD detection, which need to be determined in some way. Some studies assume a portion of OOD samples to be given and regard them as a 'validation' set, by which the hyperparemters are determined.

ODIN [3] applies perturbation with a constant magnitude $\epsilon$ to an input $\mathbf{x}$ in the direction of increasing the confidence score (i.e., the maximum softmax) and then uses the increased score in the same way as the baseline. An observation behind this procedure is that such perturbation tends to increase confidence score more for ID samples than for OOD samples. Rigorously, $\mathbf{x}$ is perturbed to increase a temperature-scaled softmax value. Thus, ODIN has two hyperparameters $\epsilon$ and the softmax temperature. In the experiments reported in [3], $\epsilon$ as well as the temperature are determined by using a portion of samples from a target OOD dataset; this is done for each pair of ID and OOD datasets.

The current state-of-the-art of OOD detection is achieved by the methods [4, 38] employing input perturbation similar to ODIN. It should be noted that there are many studies with different motivations, such as generative models [21, 55], a prior distribution [19], robustification by training networks to predict word embedding of class labels [39], pretraining of networks [7, 56], and batch-wise fine-tuning [27].

In [38], a method that employs an ensemble of networks and similar input perturbation is proposed, achieving the state-of-the-art performance. In the training step of this method, ID classes are split into two sets, one of which is virtually treated as ID classes and the other as OOD classes. A network is then trained so that the entropy for the former samples is minimized while that for the latter samples is maximized. Repeating this for different $K$ splits of classes yields $K$ leave-out classifiers (i.e., networks). At test time, an input $\mathbf{x}$ is given to these $K$ networks, whose outputs are summed to calculate ID class scores and an OOD score, where $\mathbf{x}$ is perturbed with magnitude $\epsilon$ in the direction of minimizing the entropy. In the experiments, $\epsilon$, the temperature, and additional hyperparameters are determined by selecting a particular dataset (i.e., iSUN [57]) as the OOD dataset, and OOD detection performance on different OOD datasets is evaluated.

In [4], another method is proposed, which models layer activation over ID samples with class-wise Gaussian distributions. It uses the induced Mahalanobis distances to class centroids for conducting the classification as well as OOD detection. It employs logistic regression integrating information from multiple layers and input perturbation similar to ODIN, which possesses several hyperparameters. For their determination, it is suggested to use explicit OOD samples, as in ODIN [3]. Another method is additionally suggested to avoid this potentially unrealistic assumption, which is to create adversarial examples for ID samples [58] and use them as OOD samples, determining the hyperparameters. However, even this method is not free of hyperparameters; the creation of adversarial examples needs at least one (i.e., perturbation magnitude). It is not discussed how to choose it in their paper.

**Evaluation Methods.**

Most of the recent studies employ the following evaluation method [2]. Specifying a pair of ID and OOD datasets (e.g., CIFAR-10 for ID and SVHN for OOD), it measures accuracy of distinguishing the OOD samples and ID samples. As the task is detection, appropriate metrics are used, such as accuracy at true positive rate (TPR) = 95%, area under the ROC curve (AUROC), and under the precision-recall curve (AUPR). As is noted in Sec. 2.1, most of the existing methods assume the availability of OOD samples and use them to determine their hyperparameters. Note that these OOD samples are selected from the *true* OOD dataset specified in this evaluation method. We will refer to this *one-vs-one* evaluation.

Recently, Shafaei et al. have raised a concern about the dependency of the existing methods on the explicit knowledge of the true OOD dataset, and proposed a novel evaluation method that aims at measuring the practical performance of OOD detection [1]. It assumes an ID dataset and multiple OOD datasets $\mathcal{D} = \{D_1, \ldots\}$ for evaluation. Then, the evaluation starts with choosing one dataset $D_i \in \mathcal{D}$ and use the samples from it to determine the hyperparameters of the method under evaluation; it then evaluates its detection accuracy when regarding each of the other datasets in $\mathcal{D}$ (i.e., $\mathcal{D} \backslash D_i$) as the OOD dataset, reporting the average accuracy over $\mathcal{D} \backslash D_i$. Note that this test returns the accuracy for each dataset in $\mathcal{D}$ (used for the assumed OOD dataset). We will refer to this *less-biased* evaluation.

### 2.2.3 Cosine Similarity

The proposed method employs softmax of scaled cosine similarity instead of ordinary softmax of logits. A similar approach has already been employed in recent studies of metric learning, such as $L_2$-constrained softmax [41], SphereFace [42], NormFace [43], CosFace [44], ArcFace [45], AdaCos [46], etc. Although it may seem straightforward to apply these methods to OOD detection, to the authors' knowledge, there is no study that has tried this before.

These metric learning methods are identical in that they use cosine similarity. They differ in i) if and how the weight $\mathbf{w}$ or the feature $\mathbf{f}$ of the last layer are normalized; ii) if and how margins are used with the cosine similarity to encourage maximization of inter-class variance and minimization of intra-class variance; and iii) how the scale parameter (i.e., $s$ in (2.3)) is treated, i.e., as either a hyperparameter, a learnable parameter [43], or other [46]. According to this categorization, our method is the most similar to NormFace [43] and AdaCos [46], in which both $\mathbf{w}$ and $\mathbf{f}$ are normalized and no margin is utilized. However, our method still differs from these metric learning methods in that it predicts $s$ along with class probabilities at inference time. Ours also differs in that it uses a single fully-connected layer to compute the cosine similarity, whereas these metric learning methods use two fully-connected layers.

## 2.3 Proposed Method

### 2.3.1 Softmax of Scaled Cosine Similarity

The standard formulation of multi-class classification is to make the network predict class probabilities for an input, and use cross-entropy loss to evaluate the correctness of the prediction. The predicted class probabilities are obtained by applying softmax to the linear transform $\mathbf{W}\mathbf{f}+\mathbf{b}$ of the activation or feature $\mathbf{f}$ of the last layer, and then the loss is calculated assuming 1-of-$K$ coding of the true class $c$ as

$$\mathcal{L} = -\log \frac{e^{\mathbf{w}_c^\top \mathbf{f}+b_c}}{\sum_{i=1}^{C} e^{\mathbf{w}_i^\top \mathbf{f}+b_i}}, \tag{2.1}$$

where $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_C]^\top$ and $\mathbf{b} = [b_1, \ldots, b_C]^\top$.

Metric learning attempts to learn feature space suitable for the purpose of open-set classification, e.g., face verification. Unlike earlier methods employing triplet loss [59, 60] and contrastive loss [61, 62], recent methods [43–45] modify the loss (2.1) and minimize the cross entropy loss as with the standard multi-class classification. The main idea is to use the cosine of the angle between the weight $\mathbf{w}_i$ and the feature $\mathbf{f}$ as a class score. Specifically, $\cos\theta_i \equiv \mathbf{w}_i^\top \mathbf{f}/(\|\mathbf{w}_i\|\|\mathbf{f}\|)$ is used instead of the logit $\mathbf{w}_i^\top \mathbf{f} + b_i$ in (2.1); then a new loss is given as

$$\mathcal{L} = -\log \frac{e^{\cos\theta_c}}{\sum_{i=1}^{C} e^{\cos\theta_i}}. \tag{2.2}$$

The behavior of softmax, i.e., how soft its maximum operation will be, depends on the distribution of its inputs, which can be controlled by a scaling parameter of the inputs, called temperature $T$. This parameter is used for several purposes [11,37]. In metric learning methods, it is employed to widen the range $[-1, 1]$ of $\cos\theta_i$'s inputted to softmax; specifically, all the input cosine $\cos\theta_i$'s are scaled by a parameter $s(= 1/T)$, revising the above loss as

$$\mathcal{L} = -\log \frac{e^{s\cos\theta_c}}{\sum_{i=1}^{C} e^{s\cos\theta_i}}. \tag{2.3}$$

## 2.3.2 Predicting the Scaling Parameter

In most of the metric learning methods employing similar loss functions, the scaling parameter $s$ in (2.3) is treated as either a hyperparameter chosen in a validation step or a parameter automatically determined in the training step [43,46] There is yet another method for determining $s$, which is to predict it from $\mathbf{f}$ together with class probabilities. This makes the method hyperparameter-free. Moreover, we empirically found that this performs the best. Among several ways of computing $s$ from $\mathbf{f}$, the following works the best:

$$s = \exp\{\mathrm{BN}(\mathbf{w}_s^\top \mathbf{f} + b_s)\}, \tag{2.4}$$

where BN is batch normalization [36], and $\mathbf{w}_s$ and $b_s$ are the weight and bias of the added branch to predict $s$.

## 2.3.3 Design of the Output Layer

In the aforementioned studies of metric learning, ResNets are employed as a base network and are modified to implement the softmax of cosine similarity. Modern CNNs like ResNets are usually designed to have a single fully-connected (FC) layer between the final pooling layer (i.e., global average pooling) and the network output. As ReLU activation function is applied to the inputs of the pooling layer, if we use the last FC layer for computing cosine similarity (i.e., treating its input as $\mathbf{f}$ and its weights as $\mathbf{w}_i$'s), then the elements of $\mathbf{f}$ take only non-negative values. Thus, the metric learning methods add an extra single FC layer on top of the FC layer and use the output of the first FC layer as $\mathbf{f}$, making $\mathbf{f}$ (after normalization) distribute on the whole hypersphere. In short, the metric learning methods employ two FC layers at the final section of the network.

However, we found that for the purpose of OOD detection, having two fully-connected layers does not perform better than simply using the output of the final pooling layer as $\mathbf{f}$. Details will be given in our experimental results. Note that in the case of a single FC layer, as $\mathbf{f}$ takes only non-negative values, $\mathbf{f}$ resides in the first quadrant of the space, which is very narrow subspace comparative to the entire space.

To train the modified network, we use a standard method. In our experiments, we employ SGD with weight decay as the optimizer, as in the previous studies of OOD detection [3, 4, 38, 39]. In several studies of metric learning [44, 45, 63], weight decay is also employed on all the layers of networks. However, it may have different effects on the last layer of the network employing cosine similarity, where weights are normalized and thus its length does not affect the loss. In our experiments, we found that it works better when we do not apply weight decay to the last layer.

### 2.3.4 Detecting OOD samples

Detecting OOD samples is performed in the following way. Given an input $\mathbf{x}$, our network computes $\cos\theta_i$ $(i = 1, \ldots, C)$. Let $i_{max}$ be the index of the maximum of these cosine values. We use $\cos\theta_{i_{max}}$ for distinguishing ID and OOD samples. To be specific, setting a threshold, we declare $\mathbf{x}$ is an OOD sample if $\cos\theta_{i_{max}}$ is lower than it. Otherwise, we classify $\mathbf{x}$ into the class $i_{max}$ with the predicted probability $e^{s\cos\theta_{i_{max}}} / \sum e^{s\cos\theta_i}$.

## 2.4 Experimental Results

### 2.4.1 Experimental Settings

We conducted experiments to evaluate the proposed method and compare it with existing methods.

**Evaluation Methods.**

We employ the one-vs-one and less-biased evaluation methods explained in Sec. 2.2.2. The major difference between the two is in the assumption of prior knowledge about OOD datasets, which affects the determination of the hyperparameters of the OOD detection methods under evaluation. Note therefore that *the difference does not matter for our method*, as it does not need any hyperparameter; it only affects the other compared methods.

**One-vs-one evaluation** This evaluation assumes one ID and one OOD datasets. A network is trained on the ID dataset and each method attempts to distinguish ID and OOD samples using the network. Each method may use a fixed number of samples from the specified OOD datasets for its hyperparameter determination. We followed the experimental configurations commonly employed in the previous studies [3, 4, 38].

**Less-biased evaluation** This evaluation uses one ID and many OOD datasets. Each method may access one of the OOD datasets to determine its hyperparameters but its evaluation is conducted on the task of distinguishing the ID samples and samples from each of the other OOD datasets. We followed the study of Shafaei et al. [1] with slight modifications. First, we use AUROC instead of detection accuracy for evaluation metrics (additionally, accuracy at TPR= 95% and AUPR-IN in Sec. 2.6), as we believe that they are better metrics for detection tasks, and they are employed in the one-vs-one evaluation. Second, we add more OOD datasets to those used in their study to further increase the effectiveness and practicality of the evaluation.

**Tasks and Datasets.**

We use CIFAR-10/100 for the target classification tasks in all the experiments. Using them as ID datasets, we use the following OOD datasets in one-vs-one evaluation: TinyImageNet (cropped and resized) [64], LSUN (cropped and resized) [65], iSUN [57],[1] SVHN [66] and Food-101 [67] For the less-biased evaluation, we additionally use STL-10 [68], MNIST [69], NotMNIST, and Fashion MNIST [70]. As for STL-10 and Food-101, we resize their images to $32 \times 32$ pixels.

[1]Datasets are available at `https://github.com/facebookresearch/odin`.

**Remark** We found that the cropped images of TinyImageNet and LSUN that are provided by the GitHub repository of [3], which are employed in many recent studies, have a black frame of two-pixel width around them; see Sec. 2.11 for details. Although we are not sure if this is intentional, considering that the frame will make OOD detection easier, we use two versions with/without the black frame in our experiments; the frame-free version is indicated by '$*$' in what follows. In this section, we show mainly results on the frame-free versions. Those on the original versions are shown in Sec. 2.6, although it does not affect our conclusion.

**Networks and Their Training.**

For networks, we employ the two CNNs commonly used in the previous studies, i.e., *Wide ResNet* [71] and *DenseNet* [72] as the base networks. Following [3], we use WRN-28-10 and DenseNet-100-12 having 100 layers with growth rate 12. The former is trained with batch size = 128 for 200 epochs with weight decay = 0.0005, and the latter is trained with batch size = 64 for 300 epochs with weight decay = 0.0001. Dropout is not used in the both networks. We employ a learning rate schedule, where the learning rate starts with 0.1 and decreases by 1/10 at 50% and 75% of the training steps.

The proposed method modifies the final layer and the loss of the base networks. Table 2.1 shows comparisons between the base networks and their modified version. The numbers are an average over five runs and their standard deviations are shown in parenthesis. It is seen that the modification tends to lower classification accuracy by a small amount. If this difference does matter, one may use the proposed network only for OOD detection and the standard network for ID classification.

**Compared Methods.**

The methods we compare are as follows: the baseline method [2], ODIN [3], Mahalanobis detector [4][2], and leave-out ensemble [38]. The last two methods are reported to achieve the highest performance in the case of a single network and multiple networks, respectively. We conduct experiments separately with the first

[2]We used the publicly available code: `https://github.com/pokaxpoka/deep_Mahalanobis_detector`

Table 2.1: Performance of the base networks and their modified versions for the proposed method for the task of classification of ID (in-distribution) samples.

| Network | In-Dist | Testing Accuracy | |
|---|---|---|---|
| | | Standard | Cosine |
| Dense-100-12 | CIFAR-10 | 95.11(0.10) | 94.92(0.04) |
| | CIFAR-100 | 76.97(0.24) | 75.65(0.12) |
| WRN-28-10 | CIFAR-10 | 95.99(0.09) | 95.72(0.05) |
| | CIFAR-100 | 81.04(0.37) | 78.53(0.28) |

three and the last one due to the difference in settings. We report those with the leave-out ensemble in Sec. 2.10.

All these methods (but the baseline) have hyperparameters for OOD detection. For ODIN and the Mahalanobis detector, we follow the authors' methods [3, 4] to determine them using a portion of the true OOD dataset. For the leave-out ensemble (comparisons in Sec. 2.10), we use the values of detection accuracy from its paper [38], in which the authors use a specific OOD dataset (iSUN) for hyperparameter determination.

### 2.4.2 Comparison by Less-biased Evaluation

We first show the performance of the four methods, i.e., the baseline, ODIN, the Mahalanobis detector, and ours, measured by the less-biased evaluation method. Figure 2.1 shows the results[3]. The details of the experimental settings are as follows. We use either CIFAR-10 or CIFAR-100 for the ID dataset. For the actual OOD dataset, we choose one of the eleven datasets described above and evaluate the OOD detection performance on each of the eleven ID-OOD pairs. For each ID-OOD pair, we use one of the rest (i.e., ten datasets) as a hypothesized OOD dataset, using which the hyperparameters are chosen for ODIN and Mahalanobis detector. We iterate this for the ten datasets. For each method/setting, we evaluate five models trained from

[3]A complete table including other metrics, i.e., accuracy at TPR= 95% and AUPR-IN, is shown in Sec. 2.6

different initial values. Finally, we calculate, for each method on each ID-OOD pair, the mean and standard deviation of AUROC (a bar and its error bar in Fig. 2.1) over the five models and the ten hypothesized datasets (for ODIN and Mahalanobis).

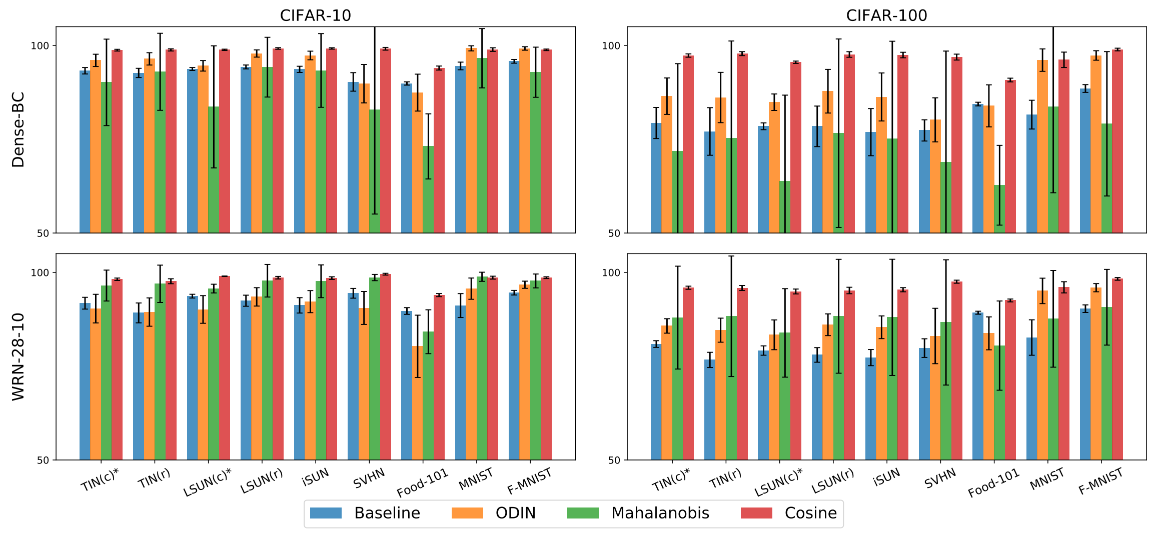


Figure 2.1: OOD detection performance (AUROC) measured by the less-biased evaluation [1] for the baseline method [2], ODIN, [3] and the Mahalanobis detector [4], and the proposed one (denoted as 'Cosine'). Other metrics, i.e., accuracy at TPR=95% and AUPR-IN, are reported in Sec. 2.6

It is seen from Fig. 2.1 that the proposed method consistently achieves better performance than others. It is noted that the Mahalanobis detector, which shows the state-of-the-art performance in the conventional (i.e., one-vs-one) evaluation, shows unstable behaviors; the mean of AUROC tends to vary significantly and the standard deviation is very large depending on the dataset used for hyperparameter determination. The same observation applies to ODIN.

This clearly demonstrates the issue with these methods, that is, their performance is dependent on the choice of the hyperparameters. On the other hand, the proposed method performs consistently for all the cases. This is also confirmed from Fig. 2.2, which shows a different plot of the same experimental result; it shows AUROC for a *single* OOD dataset instead of the *mean* over multiple OOD datasets shown in Fig. 2.1. It is seen that the performance of the Mahalanobis detector varies a lot depending on the assumed OOD dataset. Additionally, it can be seen that the dataset yielding the highest performance differs for different true OOD datasets; iSUN, TIN(r), or

Gaussian etc. is the best for detecting LSUN(r) as OOD, whereas F-MNIST or NotMNIST is the best for detecting MNIST as OOD.

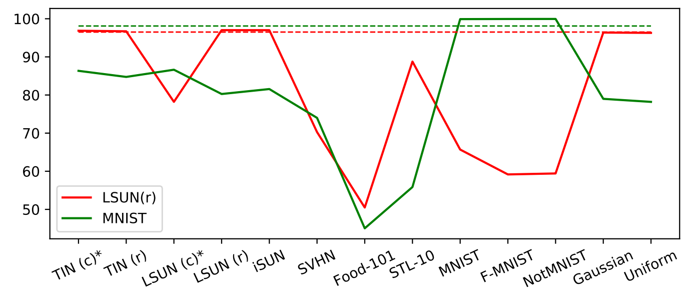


Figure 2.2: Dependency of detection performance (AUROC) on the assumed OOD datasets (whose names are given in the horizontal axis) used for determining hyperparameters. Mahalanobis detector (solid lines) [4] and our method (broken lines). CIFAR-100 is used as ID and either LSUN(r) (in red color) or MNIST (in green color) is used as true OOD. DenseNet-100-12 is used for the network. Our method does not have hyperparameters and thus is independent of the assumed OOD dataset.

### 2.4.3 Comparison by One-vs-one Evaluation

We then show the comparison of the same four methods in the one-vs-one evaluation, which is employed in the majority of the previous studies. We ran each method five times from the training step, where the network weights are initialized randomly, and report the mean and standard deviation here. Table 2.2 shows the results. It is observed that the proposed method achieves better or at least competitive performance to the others. When using DenseNet-100-12, the proposed method consistently achieves higher performance than the Mahalanobis detector on almost all the datasets.

### 2.4.4 Ablation Study

Although the proposed method employs softmax of cosine similarity equivalent to metric learning methods, there are differences in detailed designs, even compared with the most similar NormFace [43]. To be specific, they are the scale prediction (referred to as *Scale* in Table 2.3), the use of a single FC layer instead of two FC layers (*Single FC*), and non-application of weight decay to the last FC layer (*w/o*

Table 2.2: OOD detection performance of the four methods measured by conventional one-vs-one evaluation.

| | ID | OOD | AUROC | | | |
|---|---|---|---|---|---|---|
| | | | Base [2] | ODIN [3] | Maha [4] | Cosine |
| Dense-100-12 | CIFAR-10 | TIN (c) | 94.90(0.43) | 98.79(0.32) | 94.48(1.19) | **98.89(0.24)** |
| | | TIN (c)* | 93.26(0.85) | 96.67(0.97) | 97.36(0.39) | **98.74(0.23)** |
| | | TIN (r) | 92.67(1.23) | 97.20(1.17) | **98.91(0.23)** | 98.82(0.29) |
| | | LSUN (c) | 95.57(0.20) | 98.48(0.14) | 89.06(3.21) | **99.09(0.12)** |
| | | LSUN (c)* | 93.72(0.39) | 96.41(0.52) | 93.63(0.69) | **98.83(0.18)** |
| | | LSUN (r) | 94.28(0.52) | 98.43(0.49) | 99.00(0.23) | **99.19(0.22)** |
| | | iSUN | 93.62(0.83) | 97.92(0.71) | 98.95(0.21) | **99.20(0.19)** |
| | | SVHN | 90.28(2.47) | 95.11(0.48) | 98.89(0.37) | **99.11(0.36)** |
| | | Food-101 | 89.87(0.44) | 92.06(0.71) | 80.38(3.83) | **93.98(0.54)** |
| | CIFAR-100 | TIN (c) | 83.70(4.00) | 94.48(3.21) | 92.97(1.63) | **97.90(0.29)** |
| | | TIN (c)* | 79.32(4.14) | 88.54(4.27) | 93.18(0.39) | **97.31(0.45)** |
| | | TIN (r) | 77.07(6.35) | 88.14(6.92) | 96.81(0.27) | **97.82(0.53)** |
| | | LSUN (c) | 82.92(0.59) | 94.72(0.59) | 91.65(2.96) | **96.73(0.31)** |
| | | LSUN (c)* | 78.46(0.91) | 87.89(1.13) | 85.44(1.85) | **95.52(0.32)** |
| | | LSUN (r) | 78.44(5.41) | 90.38(4.76) | 97.00(0.15) | **97.59(0.75)** |
| | | iSUN | 76.89(6.28) | 88.27(6.49) | 97.04(0.10) | **97.45(0.73)** |
| | | SVHN | 77.36(2.83) | 91.60(0.73) | 96.48(0.68) | **96.90(0.79)** |
| | | Food-101 | 84.38(0.48) | **90.82(0.60)** | 67.14(1.39) | 90.79(0.49) |

| | ID | OOD | AUROC | | | |
|---|---|---|---|---|---|---|
| | | | Base [2] | ODIN [3] | Maha [4] | Cosine |
| WRN-28-10 | CIFAR-10 | TIN (c) | 93.86(0.90) | 95.88(1.01) | 95.99(1.04) | **98.35(0.32)** |
| | | TIN (c)* | 91.79(1.57) | 92.17(2.19) | **98.50(0.11)** | 98.17(0.33) |
| | | TIN (r) | 89.21(2.65) | 90.60(3.21) | **99.15(0.18)** | 97.65(0.66) |
| | | LSUN (c) | 95.41(0.26) | 97.20(0.15) | 92.65(1.33) | **99.19(0.07)** |
| | | LSUN (c)* | 93.67(0.50) | 95.08(0.42) | 96.90(0.35) | **98.98(0.07)** |
| | | LSUN (r) | 92.45(1.48) | 94.48(1.70) | **99.37(0.13)** | 98.59(0.34) |
| | | iSUN | 91.22(2.05) | 93.25(2.43) | **99.29(0.10)** | 98.48(0.36) |
| | | SVHN | 94.43(1.30) | 93.34(3.60) | 99.28(0.09) | **99.52(0.24)** |
| | | Food-101 | 89.71(0.90) | 89.18(2.37) | 90.43(1.54) | **93.95(0.41)** |
| | CIFAR-100 | TIN (c) | 84.47(1.24) | 91.72(1.10) | 92.58(2.60) | **96.76(0.34)** |
| | | TIN (c)* | 80.90(0.90) | 87.08(1.29) | **96.45(0.30)** | 95.91(0.42) |
| | | TIN (r) | 76.67(2.03) | 86.28(2.43) | **97.82(0.13)** | 95.84(0.67) |
| | | LSUN (c) | 81.91(1.31) | 91.75(0.44) | 80.48(1.14) | **96.09(0.62)** |
| | | LSUN (c)* | 79.17(1.25) | 88.06(0.46) | 91.13(0.52) | **94.92(0.65)** |
| | | LSUN (r) | 78.00(1.95) | 87.90(1.83) | **97.80(0.15)** | 95.18(0.86) |
| | | iSUN | 77.29(2.15) | 87.07(2.00) | **97.66(0.14)** | 95.39(0.55) |
| | | SVHN | 79.82(2.49) | 93.46(1.05) | **97.96(0.49)** | 97.52(0.41) |
| | | Food-101 | 89.25(0.40) | 90.76(0.35) | 91.15(0.66) | **92.53(0.38)** |

*WD*). To see their impacts on performance, we conducted an ablation study, in which WRN-28-10 is used for the base network and TinyImageNet (resized) is chosen for an OOD dataset.

Table 2.3 shows the results. Row 1 shows the results of the baseline method [2], which are obtained in our experiments. Row 2 shows the results obtained by incorporating the scale prediction in the standard networks; to be specific, $s$ predicted from $\mathbf{f}$ according to (2.4) is multipled with logits as $s \cdot (\mathbf{w}_i\mathbf{f} + b_i)$ ($i = 1, \ldots, C$), which are then normalized by softmax to yield the cross-entropy loss. As is shown in Row 2, this simple modification to the baseline boosts the performance, which is surprising.

Row 3 and below show results when cosine similarity is used for OOD detection. Rows 3 to 6 show the results obtained when a fixed value (i.e., 16, 32, 64, 128) is chosen for $s$. It is observed from this that the application of scaling affects a lot detection performance, and it tends to be sensitive to their choice. This means that, if $s$ is treated as a fixed parameter, it will become a hyperparameter that needs to be tuned for each dataset. Row 7 shows the result when the scale is predicted from $\mathbf{f}$ as in Row 1 but with cosine similarity. It is seen that this provides results comparable to the best case of manually chosen scales.

Table 2.3: Ablation tests for evaluating the contribution of different components (i.e., ‘Cosine’, ‘Single FC’, ‘Scale’, and ‘w/o WD’; see details from the main text) of the proposed method. AUROCs for detection of OOD samples (TinyImageNet (resized)) are shown.

| | Cosine | Single FC | Scale | w/o WD | C-10 | C-100 |
|---|---|---|---|---|---|---|
| (1) | | | Baseline [2] | | 89.22 | 76.59 |
| (2) | ✗ | ✓ | Pred | ✗ | 95.74 | 88.70 |
| (3) | ✓ | ✓ | 16 | ✗ | 94.09 | 82.76 |
| (4) | ✓ | ✓ | 32 | ✗ | 96.53 | 89.02 |
| (5) | ✓ | ✓ | 64 | ✗ | 87.06 | 95.66 |
| (6) | ✓ | ✓ | 128 | ✗ | 62.02 | 94.82 |
| (7) | ✓ | ✓ | Pred | ✗ | 95.16 | 91.30 |
| (8) | ✓ | ✓ | **Pred** | ✓ | **97.66** | **95.84** |
| (9) | ✓ | ✗ | Pred | ✗ | 94.71 | 87.55 |
| (10) | ✓ | ✗ | Pred | ✓ | 89.90 | 86.96 |

Row 8 shows the results obtained by further stopping application of weight decay to the last layer, which is the proposed method. It is seen that this achieves the best performance for both CIFAR-10 and CIFAR-100. Rows 9 and 10 show the results obtained by the network having two FC layers in its final part, as in the recent metric learning methods. Following the studies of metric learning, we use 512 units in the intermediate layer. In this architecture, it is better to employ weight decay in the last layer as with the metric learning methods (i.e., Rows 9 vs 10). In conclusion, these results confirm that the use of cosine similarity as well as all the three components are indispensable to achieve the best performance.

## 2.5 Effectiveness of the Scaling Factor

### 2.5.1 Explanation by Hsu et al.

Our method and that of Hsu et al. [5] share the key component, the scaled cosine similarity, $s\cos\theta_i$, in which the angle $\theta_i$ with the $i$-th class centroid as well as the scale

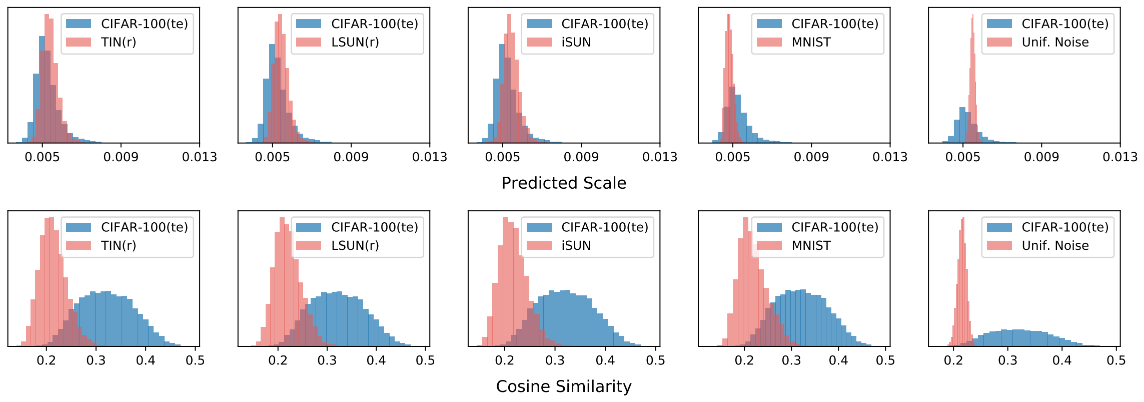


Figure 2.3: Upper: Histograms of $g(x)$ of [5] for samples from ID (CIFAR-100) and different OOD datasets, respectively. Lower: Histograms of the cosine similarity for the same ID and OOD samples. The network is WRN-28-10 and dropout is not employed.

$s$ are both *predicted* from the input $x$. In [5], not $s$ but its inverse (i.e., $1/s$), denoted by $g(x)$, is predicted in a different way. The authors argue that $g(x)$ approximates $p(d_{in}|x)$, the probability of the input $x$ being in-distribution. They then argue that this contributes to better OOD detection performance. However, this explanation contradicts with empirical observation, and therefore it must be wrong. Figure 2.3 (the upper row) shows the histograms of $g(x)$, which is computed according to the method of [5], for ID and OOD samples. Here, we use CIFAR-100 for the ID dataset and several others for OOD datasets; test samples are used for both. As is clearly seen, $g(x)$ is statistically not larger for ID samples than OOD samples, although its value should be consistently larger for ID than OOD samples if their argument is true. The lower row of Fig. 2.3 shows the histograms of the cosine similarity that is used for detecting OOD samples, showing its ability to distinguish ID and OOD samples. In short, $g(x)$ cannot be seen as an approximation of $p(d_{in}|x)$ and it alone cannot detect OOD samples with good accuracy. The authors show that using dropout regularization induces different behavior of $g(x)$ between ID and OOD samples, but it is not employed in the main experiments evaluating OOD detection performance. Although it is not clear why the use of dropout makes $g(x)$ behave (slightly) differently, it should be concluded that the aforementioned claim on $g(x)$ is not the reason for the good performance of OOD detection.

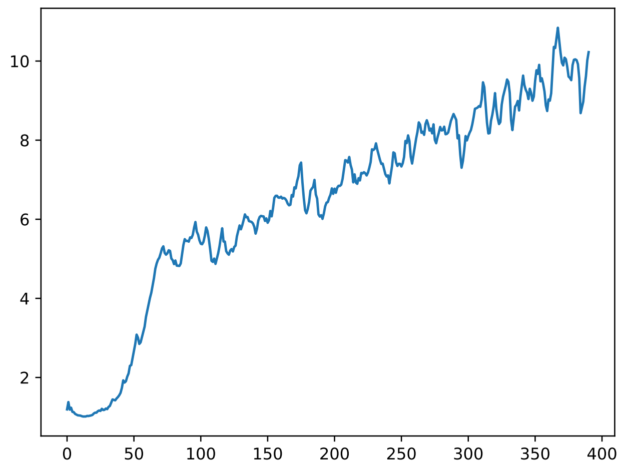


Figure 2.4: Evolution of the scale $s$ in first training epoch. The x-axis shows the training step and the y-axis indicates the average scale over the training batch.

### 2.5.2 Why Is Predicting $s$ Essential?

Then, why is it essential to make the network predict the scale $s$. We remind the readers that we use $\cos\theta_i$ without $s$ to detect OOD samples, which is the case as well with [5]. Thus, it is obviously associated not with prediction but with learning; that is, it contributes to better learning of feature space for OOD detection. An observation from our experiments is that $s$ tends to be small at the initial training stage and becomes larger as the training goes, as shown in Fig. 2.4. This is reasonable since small $s$ induces high entropy (i.e., softmax scores being more uniform and flattened) and large $s$ induces low entropy; at the initial stage, there are a lot of misclassifications due to random weight initialization, leading to large cross-entropy loss, which will be compensated by making $s$ small. More importantly, once the network has learned to correctly classify ID samples, or specifically, once it has learned to be able to consistently output max-logits for the correct classes, then $s$ will start to become large; the minimization of the loss will be achieved not by reorganizing the feature space but by making $s$ larger. We conjecture that this mechanism serves as a regularization to avoid overfitting the learned feature space to ID samples, while such overfitting occurs in the training of the standard networks. We believe that this leads to the difference in the OOD detection performance between the proposed cosine networks and standard networks.

## 2.6 Detailed Results of Less-Biased Evaluation

In Sec. 2.4.2, we report results of the less-biased evaluation in Fig. 2.1. We show here additional, more detailed results of the same experiments including two additional metrics, i.e., accuracy at TPR= 95% and AUPR-In, in Table 2.4. The details of the experimental configurations are provided in Sec. 2.4.1.

In the above experiments, we used TIN(c)* and LSUN(c)*, i.e., our corrected version of the cropped images from TinyImageNet and LSUN; see Sec. 2.11. For the sake of completeness, we also conducted the same experiments using TIN(c) and LSUN(c), their original versions having a two-pixel black frame, supplied by the GitHub repository of the authors of ODIN [3]. Table 2.5 shows the results.

## 2.7 Dependency on an OOD Validation Dataset: Full Version

In Fig. 2.2 of Sec. 2.4.2, we demonstrate the dependency of the previous methods on the assumed OOD dataset used for hyperparameter determination, where only TinyImageNet (resized) and F-MNIST are used as the assumed datasets. We show here the complete results in Fig. 2.5; it shows AUROC of detecting OOD samples given in the horizontal axis when CIFAR-100 [73] is the ID dataset and WRN-28-10 is employed.

## 2.8 Mahalanobis Detector with Hyperparameters Tuned by Adversarial Samples

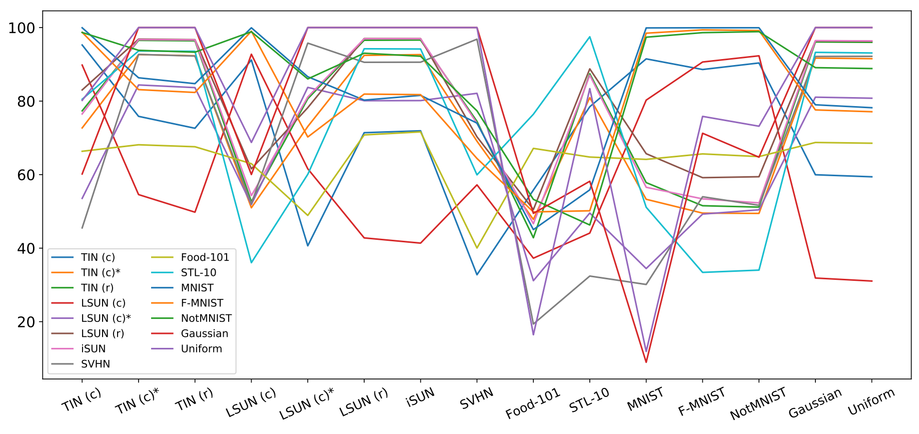
As mentioned in Sec. 2.2.2, the Mahalanobis detector [4] has two modes of selecting its hyperparameters. One is to use the explicit OOD samples and the other is to create adversarial examples from the ID samples and assume them as OOD samples. Although the latter does not need explicit OOD samples, it incorporates another hyperparameter(s) for the creation of the adversarial examples, and thus will not be an ideal solution, as we point out previously. One may wonder, however, how

Table 2.4: More detailed results of OOD detection performance measured by the less-biased evaluation. The same four methods as those considered in Sec. 2.4.2 are compared.

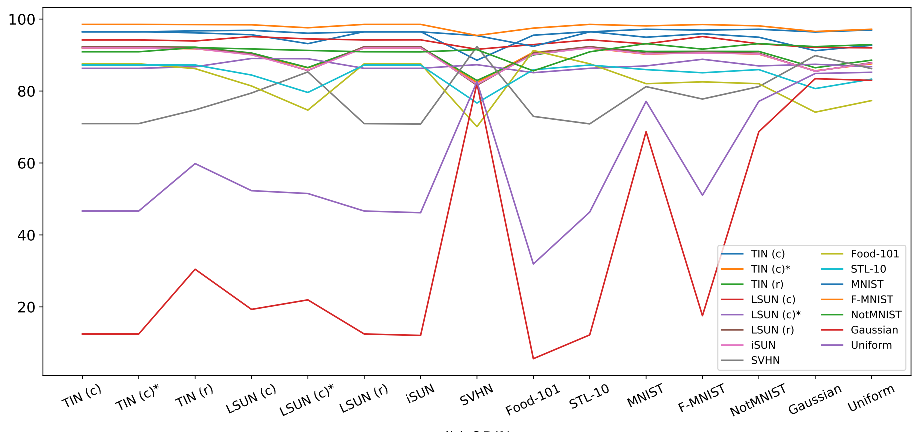
| Network | ID | OOD | Accuracy at TPR= 95% | AUROC | AUPR-In |
|---|---|---|---|---|---|
| | | | | Baseline [2] / ODIN [3] / Mahalanobis [4] / Ours | |
| Dense-100-12 | CIFAR-10 | TIN (c)* | 74.53(1.93) / 87.45(3.49) / 81.94(11.54) / **94.34(0.61)** | 93.26(0.85) / 96.02(1.64) / 90.16(11.53) / **98.74(0.23)** | 94.61(0.80) / 95.87(2.20) / 89.54(13.12) / **98.86(0.19)** |
| | | TIN (r) | 73.51(2.48) / 88.42(4.02) / 86.52(12.24) / **94.67(0.60)** | 92.67(1.23) / 96.43(1.64) / 92.97(10.27) / **98.82(0.29)** | 94.06(1.17) / 96.42(1.92) / 92.36(11.57) / **98.89(0.26)** |
| | | LSUN (c)* | 75.35(1.90) / 85.40(2.07) / 74.04(9.96) / **94.54(0.39)** | 93.72(0.39) / 94.57(1.40) / 83.64(16.27) / **98.83(0.18)** | 95.06(0.27) / 93.63(2.32) / 83.67(16.41) / **98.88(0.17)** |
| | | LSUN (r) | 76.74(1.40) / 92.05(2.55) / 87.07(12.11) / **95.72(0.56)** | 94.28(0.52) / 97.86(0.97) / 94.21(7.96) / **99.19(0.22)** | 95.60(0.44) / 97.99(1.01) / 94.22(8.49) / **99.26(0.20)** |
| | | iSUN | 75.12(2.08) / 90.60(2.99) / 86.53(12.74) / **95.62(0.51)** | 93.62(0.83) / 97.33(1.15) / 93.33(9.83) / **99.20(0.19)** | 95.48(0.66) / 97.69(1.14) / 93.70(9.85) / **99.33(0.16)** |
| | | SVHN | 68.84(2.90) / 76.81(7.84) / 81.63(17.75) / **95.36(0.87)** | 90.28(2.47) / 89.81(5.11) / 82.92(27.86) / **99.11(0.36)** | 82.90(7.23) / 76.03(11.04) / 78.05(30.89) / **97.99(0.73)** |
| | | Food-101 | 67.93(0.59) / 71.83(5.12) / 56.14(4.56) / **79.79(1.03)** | 89.87(0.44) / 87.43(4.92) / 73.10(8.68) / **93.98(0.54)** | 83.58(0.96) / 73.45(11.31) / 58.06(11.76) / **89.99(0.90)** |
| | | MNIST | 76.35(3.52) / **95.81(2.11)** / 91.93(11.67) / 95.15(1.51) | 94.54(1.02) / **99.24(0.68)** / 96.60(7.90) / 98.90(0.49) | 95.98(0.83) / **99.36(0.55)** / 97.66(5.91) / 99.06(0.42) |
| | | F-MNIST | 81.65(2.11) / **95.70(1.25)** / 79.22(9.62) / 94.83(0.71) | 95.76(0.50) / **99.20(0.47)** / 92.86(6.69) / 98.84(0.21) | 96.76(0.35) / **99.28(0.39)** / 94.15(6.49) / 98.94(0.17) |
| | | NotMNIST | 75.95(3.71) / 90.83(5.11) / 89.22(7.87) / **95.75(1.09)** | 93.72(1.49) / 97.33(1.75) / 94.99(7.17) / **99.01(0.30)** | 91.90(2.37) / 95.26(2.89) / 89.81(11.58) / **98.62(0.45)** |
| | | Gaussian | 63.72(14.12) / 90.27(16.11) / 95.02(10.42) / **97.50(0.00)** | 91.96(4.29) / 97.92(3.28) / 97.51(13.79) / **100.00(0.00)** | 95.17(2.27) / 98.71(1.98) / 98.35(9.25) / **100.00(0.00)** |
| | | Uniform | 60.28(19.27) / 82.27(21.01) / 94.94(10.75) / **97.50(0.00)** | 88.72(7.18) / 94.66(7.88) / 97.42(13.85) / **100.00(0.00)** | 93.28(4.22) / 96.61(5.07) / 98.20(9.49) / **100.00(0.00)** |
| | CIFAR-100 | TIN (c)* | 60.26(1.93) / 70.35(5.42) / 65.48(12.80) / **89.81(1.52)** | 79.32(4.14) / 86.48(4.87) / 71.81(23.34) / **97.31(0.45)** | 80.66(6.76) / 85.96(6.62) / 73.56(20.11) / **97.55(0.37)** |
| | | TIN (r) | 58.66(2.35) / 70.94(6.67) / 70.66(16.66) / **91.37(1.59)** | 77.07(6.35) / 86.12(6.70) / 75.25(25.95) / **97.82(0.53)** | 78.60(9.30) / 85.48(8.86) / 77.31(21.32) / **97.99(0.45)** |
| | | LSUN (c)* | 58.61(0.57) / 67.31(3.92) / 59.41(7.88) / **85.26(0.79)** | 78.46(0.91) / 84.87(2.21) / 63.89(22.87) / **95.52(0.32)** | 80.86(0.93) / 85.02(1.94) / 66.59(20.06) / **95.82(0.29)** |
| | | LSUN (r) | 59.32(2.47) / 72.23(7.36) / 70.16(17.28) / **90.69(2.23)** | 78.44(5.41) / 87.80(5.81) / 76.61(25.13) / **97.59(0.75)** | 80.74(6.55) / 87.99(6.59) / 79.78(19.97) / **97.83(0.65)** |
| | | iSUN | 58.37(2.76) / 70.66(7.07) / 69.86(17.11) / **90.48(2.17)** | 76.89(6.28) / 86.28(6.40) / 75.20(25.92) / **97.45(0.73)** | 80.56(7.56) / 87.43(6.90) / 79.64(19.95) / **97.88(0.56)** |
| | | SVHN | 56.40(1.82) / 60.28(8.31) / 64.99(14.82) / **88.42(2.57)** | 77.36(2.83) / 80.18(5.87) / 68.85(29.66) / **96.90(0.79)** | 66.58(4.07) / 67.51(7.64) / 60.65(31.79) / **93.95(1.33)** |
| | | Food-101 | 62.64(0.63) / 61.81(5.88) / 50.09(2.38) / **68.29(1.18)** | 84.38(0.48) / 83.91(5.60) / 62.73(10.63) / **90.79(0.49)** | 77.21(0.77) / 74.57(8.77) / 48.80(12.33) / **86.96(0.59)** |
| | | STL-10 | 57.82(1.32) / 67.22(4.44) / 64.71(13.15) / **75.20(3.96)** | 76.53(2.23) / 83.14(3.86) / 70.19(26.16) / **90.28(1.95)** | 81.56(2.67) / 85.23(3.91) / 76.73(20.01) / **92.43(1.54)** |
| | | MNIST | 62.68(2.68) / **87.24(6.78)** / 64.53(20.93) / 86.62(5.51) | 81.56(3.83) / 96.09(3.00) / 83.64(22.90) / **96.17(2.06)** | 84.50(3.62) / 96.49(2.65) / 89.05(16.34) / **96.62(1.82)** |
| | | F-MNIST | 70.15(1.95) / 90.24(3.60) / 62.67(18.30) / **94.69(1.10)** | 88.52(1.04) / 97.33(1.27) / 79.12(19.24) / **98.92(0.35)** | 90.30(0.80) / 97.54(1.07) / 84.07(15.08) / **99.02(0.31)** |
| | | NotMNIST | 59.04(1.67) / 79.64(6.86) / 71.91(16.07) / **84.44(3.09)** | 79.97(1.37) / 92.35(3.38) / 82.46(22.02) / **95.80(0.86)** | 73.28(2.54) / 87.56(4.71) / 78.13(22.69) / **93.87(1.12)** |
| | | Gaussian | 47.54(0.05) / 55.72(14.42) / 75.98(24.43) / **97.50(0.00)** | 53.73(13.50) / 61.56(31.80) / 68.81(40.93) / **99.74(0.52)** | 69.06(10.42) / 73.30(23.12) / 78.59(28.01) / **99.84(0.30)** |
| | | Uniform | 48.34(1.81) / 60.07(20.04) / 77.50(23.34) / **97.50(0.01)** | 63.59(20.05) / 75.66(20.60) / 73.64(39.00) / **99.71(0.56)** | 75.90(13.16) / 83.94(14.22) / 81.48(26.97) / **99.82(0.34)** |
| WRN-28-10 | CIFAR-10 | TIN (c)* | 76.10(1.38) / 81.36(3.70) / 89.67(7.11) / **92.63(1.26)** | 91.79(1.57) / 90.35(3.81) / 96.50(4.12) / **98.17(0.33)** | 90.94(2.91) / 87.44(5.17) / 96.28(5.14) / **98.42(0.27)** |
| | | TIN (r) | 72.35(2.13) / 79.39(4.11) / **91.47(7.96)** / 91.05(2.24) | 89.21(2.65) / 89.43(3.78) / 96.96(4.98) / **97.65(0.66)** | 87.62(4.65) / 86.83(5.01) / 96.60(6.01) / **97.94(0.54)** |
| | | LSUN (c)* | 78.31(2.05) / 81.76(2.78) / 85.97(2.24) / **94.96(0.16)** | 93.67(0.50) / 90.12(3.69) / 95.69(1.16) / **98.98(0.07)** | 93.90(0.56) / 86.47(5.93) / 95.98(1.79) / **99.06(0.05)** |
| | | LSUN (r) | 76.39(2.41) / 84.92(3.58) / 92.85(7.86) / **94.02(1.29)** | 92.45(1.48) / 93.48(2.45) / 97.78(4.34) / **98.59(0.34)** | 92.33(2.31) / 92.19(3.13) / 97.94(4.23) / **98.79(0.28)** |
| | | iSUN | 74.82(2.19) / 83.11(3.82) / 92.54(7.77) / **93.47(1.35)** | 91.22(2.05) / 92.21(2.94) / 97.64(4.37) / **98.48(0.36)** | 91.40(3.31) / 91.44(3.60) / 97.89(4.47) / **98.82(0.26)** |
| | | SVHN | 80.66(2.52) / 81.60(5.57) / 94.14(1.97) / **96.50(0.69)** | 94.43(1.30) / 90.52(4.39) / 98.63(0.83) / **99.52(0.24)** | 88.55(4.75) / 74.03(11.15) / 97.11(2.52) / **98.94(0.52)** |
| | | Food-101 | 71.61(1.04) / 70.21(5.85) / 63.31(5.20) / **78.73(1.42)** | 89.71(0.90) / 80.29(8.31) / 84.19(5.85) / **93.95(0.41)** | 76.04(3.29) / 55.33(13.49) / 73.64(9.95) / **90.41(0.47)** |
| | | MNIST | 72.35(5.94) / 87.91(6.39) / **95.08(4.88)** / 94.60(1.22) | 91.16(3.19) / 95.67(2.84) / **98.83(1.22)** / 98.60(0.39) | 91.37(3.95) / 95.22(3.27) / **99.19(0.86)** / 98.91(0.28) |
| | | F-MNIST | 81.20(1.34) / 90.65(2.12) / 92.05(5.25) / **94.03(0.72)** | 94.59(0.61) / 96.71(0.96) / 97.73(1.85) / **98.60(0.24)** | 94.91(0.79) / 96.13(1.20) / 98.21(1.85) / **98.78(0.20)** |
| | | NotMNIST | 81.27(2.34) / 92.38(3.23) / 95.95(2.67) / **96.33(0.62)** | 95.00(0.58) / 97.60(1.06) / **99.28(0.85)** / 99.11(0.28) | 92.37(1.43) / 94.82(1.96) / **98.85(1.40)** / 98.84(0.32) |
| | | Gaussian | 82.94(17.44) / 93.87(7.97) / 97.12(2.80) / **97.50(0.00)** | 95.16(6.00) / 98.52(2.71) / 99.60(2.91) / **99.98(0.02)** | 96.47(4.70) / 98.80(2.29) / 99.39(4.51) / **99.99(0.01)** |
| | | Uniform | 78.59(14.28) / 93.38(5.71) / 96.25(6.97) / **97.50(0.00)** | 95.30(2.90) / 98.52(1.69) / 98.01(13.34) / **99.99(0.01)** | 96.70(2.10) / 98.81(1.38) / 98.47(9.15) / **99.99(0.01)** |
| | CIFAR-100 | TIN (c)* | 61.96(0.78) / 69.18(2.82) / 77.40(12.03) / **85.01(1.35)** | 80.90(0.90) / 85.73(1.93) / 87.95(13.71) / **95.91(0.42)** | 81.38(1.79) / 85.44(2.16) / 88.31(13.86) / **96.48(0.31)** |
| | | TIN (r) | 58.89(0.99) / 68.94(4.03) / 80.24(14.04) / **84.94(2.40)** | 76.67(2.03) / 84.60(3.21) / 88.31(16.07) / **95.84(0.67)** | 76.74(3.62) / 84.20(3.69) / 88.70(15.65) / **96.40(0.47)** |
| | | LSUN (c)* | 58.99(0.77) / 66.46(5.62) / 69.79(7.98) / **82.97(1.77)** | 79.17(1.25) / 83.37(3.97) / 83.87(11.80) / **94.92(0.65)** | 80.88(1.99) / 82.91(4.16) / 84.43(12.89) / **95.45(0.57)** |
| | | LSUN (r) | 58.98(1.47) / 69.64(4.44) / 79.14(14.57) / **82.76(2.40)** | 78.00(1.95) / 86.05(2.90) / 88.30(15.18) / **95.18(0.86)** | 79.25(2.28) / 86.09(2.87) / 88.98(14.72) / **95.92(0.68)** |
| | | iSUN | 58.54(1.43) / 68.94(4.43) / 79.29(14.03) / **83.73(1.92)** | 77.29(2.15) / 85.38(3.03) / 88.01(15.49) / **95.39(0.55)** | 80.27(2.59) / 86.73(2.80) / 89.40(14.30) / **96.40(0.35)** |
| | | SVHN | 58.13(2.90) / 66.73(10.20) / 76.30(14.18) / **90.28(1.57)** | 79.82(2.49) / 83.05(7.38) / 86.67(16.70) / **97.52(0.41)** | 66.44(4.78) / 68.06(11.64) / 79.01(22.03) / **95.13(0.54)** |
| | | Food-101 | 69.12(0.76) / 63.20(4.76) / 58.46(6.74) / **72.68(1.01)** | 89.25(0.40) / 83.76(4.39) / 80.49(11.90) / **92.53(0.38)** | 83.20(0.70) / 70.63(8.14) / 73.20(16.92) / **89.37(0.54)** |
| | | STL-10 | 58.23(0.55) / 64.30(2.64) / **72.25(11.96)** / 71.03(1.17) | 77.68(0.61) / 81.20(2.27) / 84.18(16.28) / **88.89(0.51)** | 81.66(1.01) / 83.70(2.31) / 87.95(12.62) / **91.70(0.43)** |
| | | MNIST | 61.79(5.62) / 83.33(7.90) / 72.71(13.36) / **85.60(3.96)** | 82.64(4.73) / 95.07(3.39) / 87.62(12.89) / **96.03(1.49)** | 85.99(3.87) / 95.84(2.82) / 89.95(11.51) / **96.59(1.25)** |
| | | F-MNIST | 72.39(2.13) / 86.47(3.21) / 76.60(10.40) / **92.48(1.22)** | 90.33(1.02) / 95.93(1.09) / 90.71(10.09) / **98.27(0.36)** | 91.74(0.87) / 96.18(0.97) / 92.33(8.75) / **98.46(0.31)** |
| | | NotMNIST | 56.15(1.61) / 76.13(7.89) / **82.57(14.47)** / 77.23(2.62) | 80.00(2.79) / 91.23(3.99) / 89.53(17.24) / **93.05(1.47)** | 73.99(4.65) / 86.76(5.07) / 85.80(21.25) / **90.53(2.00)** |
| | | Gaussian | 47.54(0.07) / 75.96(21.36) / 88.34(18.59) / **97.50(0.00)** | 59.59(30.02) / 82.89(24.64) / 88.04(30.26) / **99.82(0.15)** | 73.02(23.16) / 88.04(17.98) / 91.67(20.57) / **99.88(0.09)** |

Table 2.5: More detailed results of OOD detection performance measured by the less-biased evaluation. The original cropped images are used instead of their '*' version.

| Network | ID | OOD | Accuracy at TPR= 95% | AUROC | AUPR-In |
|---|---|---|---|---|---|
| | | | | Baseline [2] / ODIN [3] / Mahalanobis [4] / Ours | |
| Dense-100-12 | CIFAR-10 | TIN (c) | 78.86(1.37) / 93.31(2.04) / 75.77(7.50) / **94.83(0.68)** | 94.90(0.43) / 98.31(0.82) / 86.44(8.15) / **98.89(0.24)** | 96.09(0.36) / 98.39(0.90) / 85.19(9.46) / **98.97(0.20)** |
| | | TIN (r) | 73.51(2.48) / 88.47(3.99) / 82.87(13.81) / **94.67(0.60)** | 92.67(1.23) / 96.44(1.63) / 90.45(11.13) / **98.82(0.29)** | 94.06(1.17) / 96.42(1.92) / 89.51(12.38) / **98.89(0.26)** |
| | | LSUN (c) | 81.22(1.46) / 92.22(1.49) / 65.60(6.43) / **95.27(0.19)** | 95.57(0.20) / 97.78(0.77) / 76.89(9.64) / **99.09(0.12)** | 96.59(0.11) / 97.62(1.11) / 75.44(10.64) / **99.12(0.11)** |
| | | LSUN (r) | 76.74(1.40) / 92.09(2.51) / 83.55(13.63) / **95.72(0.56)** | 94.28(0.52) / 97.87(0.96) / 92.18(8.81) / **99.19(0.22)** | 95.60(0.44) / 97.99(1.00) / 92.19(9.12) / **99.26(0.20)** |
| | | iSUN | 75.12(2.08) / 90.64(2.95) / 82.53(14.57) / **95.62(0.51)** | 93.62(0.83) / 97.34(1.14) / 90.59(11.06) / **99.20(0.19)** | 95.48(0.66) / 97.69(1.14) / 91.08(10.81) / **99.33(0.16)** |
| | | SVHN | 68.84(2.90) / 76.56(8.01) / 74.45(20.13) / **95.36(0.87)** | 90.28(2.47) / 89.62(5.30) / 72.18(32.66) / **99.11(0.36)** | 82.90(7.23) / 75.67(11.31) / 65.64(35.76) / **97.99(0.73)** |
| | | Food-101 | 67.93(0.59) / 71.93(5.11) / 56.95(4.79) / **79.79(1.03)** | 89.87(0.44) / 87.48(4.91) / 73.17(8.51) / **93.98(0.54)** | 83.58(0.96) / 73.56(11.24) / 57.03(11.25) / **89.99(0.90)** |
| | | MNIST | 76.35(3.52) / **95.86(2.06)** / 91.42(11.63) / 95.15(1.51) | 94.54(1.02) / **99.26(0.67)** / 96.59(7.93) / 98.90(0.49) | 95.98(0.83) / **99.37(0.53)** / 97.60(5.92) / 99.06(0.42) |
| | | F-MNIST | 81.65(2.11) / **95.72(1.25)** / 79.24(10.83) / 94.83(0.71) | 95.76(0.50) / **99.20(0.47)** / 92.32(7.38) / 98.84(0.21) | 96.76(0.35) / **99.29(0.39)** / 93.38(7.38) / 98.94(0.17) |
| | | NotMNIST | 75.95(3.71) / 90.85(5.01) / 88.61(9.04) / **95.75(1.09)** | 93.72(1.49) / 97.33(1.72) / 94.06(8.89) / **99.01(0.30)** | 91.90(2.37) / 95.24(2.86) / 88.25(14.73) / **98.62(0.45)** |
| | | Gaussian | 63.72(14.12) / 90.36(15.81) / 93.22(13.48) / **97.50(0.00)** | 91.96(4.29) / 97.95(3.19) / 96.12(15.44) / **100.00(0.00)** | 95.17(2.27) / 98.73(1.92) / 97.43(10.34) / **100.00(0.00)** |
| | | Uniform | 60.28(19.27) / 81.82(21.23) / 92.92(13.69) / **97.50(0.00)** | 88.72(7.18) / 94.63(7.73) / 96.62(14.15) / **100.00(0.00)** | 93.28(4.22) / 96.59(4.96) / 97.67(9.68) / **100.00(0.00)** |
| | CIFAR-100 | TIN (c) | 64.67(2.28) / 81.15(6.13) / 66.07(10.16) / **91.71(0.85)** | 83.70(4.00) / 92.79(3.90) / 73.92(19.34) / **97.90(0.29)** | 84.98(6.13) / 92.52(5.05) / 73.78(15.87) / **98.03(0.24)** |
| | | TIN (r) | 58.66(2.35) / 71.32(6.55) / 68.00(16.07) / **91.37(1.59)** | 77.07(6.35) / 86.35(6.63) / 71.75(25.16) / **97.82(0.53)** | 78.60(9.30) / 85.68(8.80) / 73.14(20.57) / **97.99(0.45)** |
| | | LSUN (c) | 62.53(0.41) / 79.02(3.86) / 60.97(11.23) / **88.48(0.77)** | 82.92(0.59) / 92.58(1.87) / 66.52(19.93) / **96.73(0.31)** | 85.41(0.77) / 92.98(1.75) / 67.46(18.44) / **96.89(0.28)** |
| | | LSUN (r) | 59.32(2.47) / 72.75(7.20) / 67.95(16.61) / **90.69(2.23)** | 78.44(5.41) / 88.13(5.65) / 73.90(24.51) / **97.59(0.75)** | 80.74(6.55) / 88.29(6.42) / 76.66(19.51) / **97.83(0.65)** |
| | | iSUN | 58.37(2.76) / 71.08(6.98) / 67.10(16.53) / **90.48(2.17)** | 76.89(6.28) / 86.56(6.29) / 71.71(25.16) / **97.45(0.73)** | 80.56(7.56) / 87.67(6.78) / 76.00(19.33) / **97.88(0.56)** |
| | | SVHN | 56.40(1.82) / 59.66(7.63) / 60.25(13.81) / **88.42(2.57)** | 77.36(2.83) / 79.74(5.50) / 61.99(27.81) / **96.90(0.79)** | 66.58(4.07) / 66.79(7.23) / 50.78(29.99) / **93.95(1.33)** |
| | | Food-101 | 62.64(0.63) / 62.29(5.44) / 50.50(2.39) / **68.29(1.18)** | 84.38(0.48) / 84.41(5.15) / 61.97(10.18) / **90.79(0.49)** | 77.21(0.77) / 75.20(8.23) / 46.60(11.77) / **86.96(0.59)** |
| | | STL-10 | 57.82(1.32) / 67.57(4.27) / 64.44(12.97) / **75.20(3.96)** | 76.53(2.23) / 83.43(3.67) / 67.71(25.77) / **90.28(1.95)** | 81.56(2.67) / 85.48(3.72) / 73.69(19.52) / **92.43(1.54)** |
| | | MNIST | 62.68(2.68) / **87.43(6.85)** / 71.45(22.63) / 86.62(5.51) | 81.56(3.83) / 96.16(3.01) / 85.24(23.54) / **96.17(2.06)** | 84.50(3.62) / 96.55(2.66) / 89.93(16.70) / **96.62(1.82)** |
| | | F-MNIST | 70.15(1.95) / 90.46(3.67) / 67.74(19.52) / **94.69(1.10)** | 88.52(1.04) / 97.41(1.28) / 80.36(20.33) / **98.92(0.35)** | 90.30(0.80) / 97.61(1.09) / 84.05(16.58) / **99.02(0.31)** |
| | | NotMNIST | 59.04(1.67) / 79.54(6.79) / 75.28(16.97) / **84.44(3.09)** | 79.97(1.37) / 92.29(3.37) / 83.24(22.35) / **95.80(0.86)** | 73.28(2.54) / 87.43(4.71) / 78.33(22.92) / **93.87(1.12)** |
| | | Gaussian | 47.54(0.05) / 54.60(13.58) / 70.35(24.18) / **97.50(0.00)** | 53.73(13.50) / 59.75(31.22) / 62.78(41.05) / **99.74(0.52)** | 69.06(10.42) / 72.10(22.75) / 74.21(27.99) / **99.84(0.30)** |
| | | Uniform | 48.34(1.81) / 59.13(19.58) / 72.29(23.29) / **97.50(0.01)** | 63.59(20.05) / 74.60(20.42) / 68.71(38.75) / **99.71(0.56)** | 75.90(13.16) / 83.24(14.11) / 77.56(27.12) / **99.82(0.34)** |
| WRN-28-10 | CIFAR-10 | TIN (c) | 78.71(1.31) / 86.55(2.87) / 78.05(7.26) / **93.26(0.96)** | 93.86(0.90) / 95.03(1.74) / 90.53(5.96) / **98.35(0.32)** | 94.45(1.23) / 94.50(2.05) / 90.09(6.58) / **98.53(0.27)** |
| | | TIN (r) | 72.35(2.13) / 79.50(4.13) / 90.84(7.94) / **91.05(2.24)** | 89.21(2.65) / 89.43(3.78) / 96.62(4.95) / **97.65(0.66)** | 87.62(4.65) / 86.82(4.98) / 96.15(5.98) / **97.94(0.54)** |
| | | LSUN (c) | 82.07(1.93) / 87.51(1.97) / 69.63(5.03) / **95.58(0.21)** | 95.41(0.26) / 94.92(1.72) / 85.47(5.73) / **99.19(0.07)** | 96.19(0.18) / 93.83(2.66) / 85.34(7.15) / **99.23(0.07)** |
| | | LSUN (r) | 76.39(2.41) / 85.04(3.58) / 92.36(7.84) / **94.02(1.29)** | 92.45(1.48) / 93.49(2.45) / 97.58(4.32) / **98.59(0.34)** | 92.33(2.31) / 92.19(3.12) / 97.72(4.22) / **98.79(0.28)** |
| | | iSUN | 74.82(2.19) / 83.22(3.82) / 91.92(7.76) / **93.47(1.35)** | 91.22(2.05) / 92.22(2.94) / 97.38(4.35) / **98.48(0.36)** | 91.40(3.31) / 91.44(3.58) / 97.62(4.45) / **98.82(0.26)** |
| | | SVHN | 80.66(2.52) / 81.36(5.60) / 93.77(2.04) / **96.50(0.69)** | 94.43(1.30) / 90.31(4.41) / 98.47(0.86) / **99.52(0.24)** | 88.55(4.75) / 73.47(11.23) / 96.80(2.55) / **98.94(0.52)** |
| | | Food-101 | 71.61(1.04) / 70.11(5.79) / 63.73(5.36) / **78.73(1.42)** | 89.71(0.90) / 80.07(8.16) / 84.27(5.84) / **93.95(0.41)** | 76.04(3.29) / 54.80(13.10) / 73.44(9.85) / **90.41(0.47)** |
| | | MNIST | 72.35(5.94) / 88.24(5.90) / **94.88(5.09)** / 94.60(1.22) | 91.16(3.19) / 95.80(2.62) / **98.77(1.33)** / 98.60(0.39) | 91.37(3.95) / 95.36(3.02) / **99.13(0.98)** / 98.91(0.28) |
| | | F-MNIST | 81.20(1.34) / 90.75(2.10) / 91.51(5.27) / **94.03(0.72)** | 94.59(0.61) / 96.73(0.96) / 97.54(1.86) / **98.60(0.24)** | 94.91(0.79) / 96.15(1.20) / 98.01(1.88) / **98.78(0.20)** |
| | | NotMNIST | 81.27(2.34) / 92.55(3.10) / 95.61(2.86) / **96.33(0.62)** | 95.00(0.58) / 97.66(1.01) / 99.08(1.08) / **99.11(0.28)** | 92.37(1.43) / 94.90(1.86) / 98.39(2.27) / **98.84(0.32)** |
| | | Gaussian | 82.94(17.44) / 94.05(7.44) / 97.12(2.80) / **97.50(0.00)** | 95.16(6.00) / 98.61(2.50) / 99.60(2.91) / **99.98(0.02)** | 96.47(4.70) / 98.87(2.11) / 99.39(4.51) / **99.99(0.01)** |
| | | Uniform | 78.59(14.28) / 93.77(5.26) / 96.25(6.97) / **97.50(0.00)** | 95.30(2.90) / 98.61(1.58) / 98.01(13.34) / **99.99(0.01)** | 96.70(2.10) / 98.87(1.30) / 98.47(9.15) / **99.99(0.01)** |
| | CIFAR-100 | TIN (c) | 65.24(1.62) / 75.01(3.51) / 67.17(8.92) / **87.94(1.16)** | 84.47(1.24) / 90.67(1.73) / 81.80(11.04) / **96.76(0.34)** | 86.08(1.27) / 91.31(1.49) / 81.95(11.95) / **97.11(0.25)** |
| | | TIN (r) | 58.89(0.99) / 68.95(3.98) / 79.33(13.77) / **84.94(2.40)** | 76.67(2.03) / 84.62(3.16) / 87.65(15.93) / **95.84(0.67)** | 76.74(3.62) / 84.22(3.63) / 87.92(15.56) / **96.40(0.47)** |
| | | LSUN (c) | 60.99(1.01) / 71.10(5.89) / 59.21(5.80) / **86.49(1.74)** | 81.91(1.31) / 88.32(3.51) / 74.40(5.23) / **96.09(0.62)** | 84.68(1.45) / 89.14(3.00) / 74.50(5.80) / **96.41(0.56)** |
| | | LSUN (r) | 58.98(1.47) / 69.67(4.41) / 78.24(14.33) / **82.76(2.40)** | 78.00(1.95) / 86.07(2.87) / 87.56(15.15) / **95.18(0.86)** | 79.25(2.28) / 86.11(2.84) / 88.12(14.79) / **95.92(0.68)** |
| | | iSUN | 58.54(1.43) / 68.97(4.40) / 78.27(13.79) / **83.73(1.92)** | 77.29(2.15) / 85.40(3.01) / 87.20(15.42) / **95.39(0.55)** | 80.27(2.59) / 86.75(2.78) / 88.57(14.29) / **96.40(0.35)** |
| | | SVHN | 58.13(2.90) / 66.89(10.04) / 73.78(15.04) / **90.28(1.57)** | 79.82(2.49) / 83.23(7.27) / 83.92(18.34) / **97.52(0.41)** | 66.44(4.78) / 68.31(11.47) / 74.78(24.30) / **95.13(0.54)** |
| | | Food-101 | 69.12(0.76) / 63.17(4.72) / 58.03(6.90) / **72.68(1.01)** | 89.25(0.40) / 83.72(4.37) / 79.23(12.65) / **92.53(0.38)** | 83.20(0.70) / 70.54(8.15) / 71.23(18.04) / **89.37(0.54)** |
| | | STL-10 | 58.23(0.55) / 64.29(2.56) / **71.78(11.79)** / 71.03(1.17) | 77.68(0.61) / 81.19(2.19) / 83.70(16.12) / **88.89(0.51)** | 81.66(1.01) / 83.70(2.23) / 87.45(12.48) / **91.70(0.43)** |
| | | MNIST | 61.79(5.62) / 83.44(7.86) / 72.43(13.40) / **85.60(3.96)** | 82.64(4.73) / 95.11(3.39) / 87.26(12.91) / **96.03(1.49)** | 85.99(3.87) / 95.87(2.82) / 89.60(11.51) / **96.59(1.25)** |
| | | F-MNIST | 72.39(2.13) / 86.51(3.21) / 75.80(10.71) / **92.48(1.22)** | 90.33(1.02) / 95.94(1.09) / 90.00(10.31) / **98.27(0.36)** | 91.74(0.87) / 96.19(0.97) / 91.63(8.96) / **98.46(0.31)** |
| | | NotMNIST | 56.15(1.61) / 76.13(7.88) / **80.87(15.35)** / 77.23(2.62) | 80.00(2.79) / 91.24(4.00) / 87.56(18.87) / **93.05(1.47)** | 73.99(4.65) / 86.78(5.08) / 83.07(23.07) / **90.53(2.00)** |
| | | Gaussian | 47.54(0.07) / 75.81(21.68) / 87.95(18.57) / **97.50(0.00)** | 59.59(30.02) / 83.00(24.41) / 87.92(30.23) / **99.82(0.15)** | 73.02(23.16) / 88.14(17.78) / 91.59(20.54) / **99.88(0.09)** |
| | | Uniform | 47.51(0.02) / 68.50(20.65) / 88.96(17.14) / **97.50(0.00)** | 60.88(14.02) / 88.19(12.01) / 88.33(29.70) / **99.95(0.10)** | 74.03(12.70) / 92.15(8.43) / 91.62(20.71) / **99.97(0.07)** |



(a) Mahalanobis Detector



(b) ODIN

Figure 2.5: Dependency of (a) Mahalanobis Detector and (b) ODIN on the assumed OOD dataset used for hyperparameter determination. AUROC of detecting OOD samples given in the horizontal axis. The line colors indicate the assumed OOD dataset.

Table 2.6: OOD detection performance of the Mahalanobis Detector [4] using the adversarial samples for its hyperparameters tuning. The results are averaged from 5 runs.

| | ID | OOD | AUROC |
|---|---|---|---|
| Dense-100-12 | CIFAR-10 | TIN (c) | 87.77(3.71) |
| | | TIN (c)* | 96.84(0.89) |
| | | TIN (r) | 98.62(0.41) |
| | | LSUN (c) | 73.16(3.51) |
| | | LSUN (c)* | 91.49(1.73) |
| | | LSUN (r) | 98.79(0.37) |
| | | iSUN | 98.72(0.37) |
| | | SVHN | 97.21(1.84) |
| | | Food-101 | 77.22(4.29) |
| | | MNIST | 98.51(0.26) |
| | | F-MNIST | 93.83(1.14) |
| | | NotMNIST | 96.60(3.07) |
| | | Gaus. Noise | 100.0(0.0) |
| | | Unif. Noise | 100.0(0.0) |
| | CIFAR-100 | TIN (c) | 79.75(3.69) |
| | | TIN (c)* | 92.70(0.39) |
| | | TIN (r) | 96.45(0.30) |
| | | LSUN (c) | 62.37(8.15) |
| | | LSUN (c)* | 84.24(2.72) |
| | | LSUN (r) | 96.66(0.29) |
| | | iSUN | 96.64(0.38) |
| | | SVHN | 93.16(1.16) |
| | | Food-101 | 72.54(1.12) |
| | | STL-10 | 93.19(1.15) |
| | | MNIST | 87.82(8.88) |
| | | F-MNIST | 80.33(3.47) |
| | | NotMNIST | 91.61(3.05) |
| | | Gaus. Noise | 100.0(0.0) |
| | | Unif. Noise | 100.0(0.0) |

| | ID | OOD | AUROC |
|---|---|---|---|
| WRN-28-10 | CIFAR-10 | TIN (c) | 87.79(1.48) |
| | | TIN (c)* | 97.54(0.53) |
| | | TIN (r) | 98.65(0.29) |
| | | LSUN (c) | 70.97(1.90) |
| | | LSUN (c)* | 93.53(0.91) |
| | | LSUN (r) | 99.16(0.21) |
| | | iSUN | 99.07(0.21) |
| | | SVHN | 98.03(0.55) |
| | | Food-101 | 73.55(6.09) |
| | | MNIST | 98.89(0.72) |
| | | F-MNIST | 97.89(0.59) |
| | | NotMNIST | 99.41(0.28) |
| | | Gaus. Noise | 100.0(0.0) |
| | | Unif. Noise | 100.0(0.0) |
| | CIFAR-100 | TIN (c) | 86.61(1.13) |
| | | TIN (c)* | 96.24(0.13) |
| | | TIN (r) | 97.69(0.19) |
| | | LSUN (c) | 67.07(3.41) |
| | | LSUN (c)* | 87.00(1.34) |
| | | LSUN (r) | 97.67(0.23) |
| | | iSUN | 97.62(0.19) |
| | | SVHN | 92.95(1.03) |
| | | Food-101 | 86.78(1.39) |
| | | STL-10 | 94.73(0.59) |
| | | MNIST | 84.63(7.77) |
| | | F-MNIST | 93.41(2.11) |
| | | NotMNIST | 97.37(0.82) |
| | | Gaus. Noise | 100.0(0.0) |
| | | Unif. Noise | 100.0(0.0) |

is the performance on this scenario. As there is no mention of the intuition of its hyperparameters choosing, we follow the one provided by the authors of [4].[4] We show its results in Table 2.6.

As shown in the table, the results show similar instability to the case of using explicit OOD samples; the method shows good performance with some OOD datasets but not with others. This is illustrated more clearly illustrated in Fig. 2.6. We suspect that it performs well only if the created adversarial examples resemble the true OOD examples. We cannot expect this is always the case in the real world problems. In short, our conclusion remains the same with this method.

[4]https://github.com/pokaxpoka/deep_Mahalanobis_detector

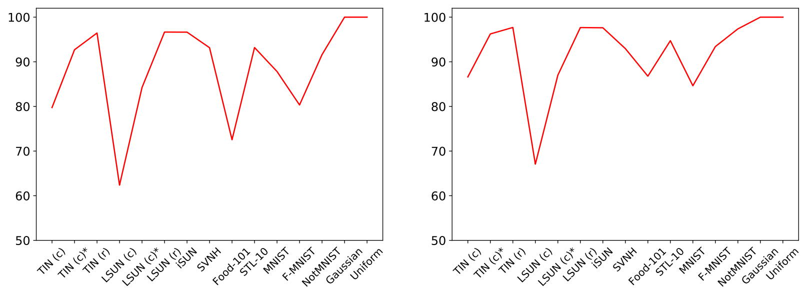


Figure 2.6: The instability of the OOD detection performance for Mahalanobis Detector using adversarial samples in its hyperparameters tuning (ID = CIFAR-100).

## 2.9 More Complete Results of One-vs-one Evaluation

Table 2.7 shows a more complete version of Table 2.2 in Sec. 2.4.3. It shows the performances measured by the two additional metrics as above.

## 2.10 OOD Detection Using an Ensemble of Networks

The leave-out ensemble proposed in [38] uses multiple networks and is reported to achieve high OOD detection performance in the one-vs-one evaluation. To make a fair comparison, we consider an extension of our method to an ensemble model. The underlying thought is that the use of an ensemble of multiple models will yield better results, as seen in many inference tasks. To be specific, in the training step, we train multiple networks on the target classification task; in our experiments, we trained models of the same architecture initialized with different random weights. At test time, given an input sample, we make the networks output the cosine similarities and calculate their averages over different networks. Table 2.8 - 2.9 show the results. For the leave-out ensemble, it shows the performances reported in [38] and those obtained in our own experiments (indicated by *); we used a public code[5] suggested by the author of [38]. It is seen that our method shows better or at least comparable performance as compared with the leave-out ensemble, even in the one-vs-one evalua-

[5]https://github.com/YU1ut/Ensemble-of-Leave-out-Classifiers

Table 2.7: Performance of four out-of-distribution detection methods on a single network using one-vs-one evaluation.

| Network | ID | OOD | Accuracy at TPR= 95% | AUROC | AUPR-In |
|---|---|---|---|---|---|
| | | | | Baseline [2] / ODIN [3] / Mahalanobis [4] / Ours | |
| Dense-100-12 | CIFAR-10 | TIN (c) | 78.86(1.37) / 94.59(0.80) / 85.51(2.28) / **94.83(0.68)** | 94.90(0.43) / 98.79(0.32) / 94.48(1.19) / **98.89(0.24)** | 96.09(0.36) / 98.86(0.28) / 93.76(1.67) / **98.97(0.20)** |
| | | TIN (c)* | 74.53(1.93) / 88.97(2.54) / 90.84(0.96) / **94.34(0.61)** | 93.26(0.85) / 96.67(0.97) / 97.36(0.39) / **98.74(0.23)** | 94.61(0.80) / 96.64(0.96) / 97.64(0.37) / **98.86(0.19)** |
| | | TIN (r) | 73.51(2.48) / 90.39(2.98) / **95.00(0.54)** / 94.67(0.60) | 92.67(1.23) / 97.20(1.17) / **98.91(0.23)** / 98.82(0.29) | 94.06(1.17) / 97.27(1.10) / **99.04(0.20)** / 98.89(0.26) |
| | | LSUN (c) | 81.22(1.46) / 93.67(0.53) / 74.91(5.87) / **95.27(0.19)** | 95.57(0.20) / 98.48(0.14) / 89.06(3.21) / **99.09(0.12)** | 96.59(0.11) / 98.62(0.16) / 89.00(2.32) / **99.12(0.11)** |
| | | LSUN (c)* | 75.35(1.90) / 87.61(1.40) / 82.51(1.62) / **94.54(0.39)** | 93.72(0.39) / 96.41(0.52) / 93.63(0.69) / **98.83(0.18)** | 95.06(0.27) / 96.69(0.53) / 94.21(0.58) / **98.88(0.17)** |
| | | LSUN (r) | 76.74(1.40) / 93.63(1.37) / 95.48(0.43) / **95.72(0.56)** | 94.28(0.52) / 98.43(0.49) / 99.00(0.23) / **99.19(0.22)** | 95.60(0.44) / 98.52(0.44) / 99.15(0.20) / **99.26(0.20)** |
| | | iSUN | 75.12(2.08) / 92.26(1.99) / 95.09(0.51) / **95.62(0.51)** | 93.62(0.83) / 97.92(0.71) / 98.95(0.21) / **99.20(0.19)** | 95.48(0.66) / 98.21(0.58) / 99.22(0.15) / **99.33(0.16)** |
| | | SVHN | 68.84(2.90) / 88.83(0.28) / 95.01(0.78) / **95.36(0.87)** | 90.28(2.47) / 95.11(0.48) / 98.89(0.37) / **99.11(0.36)** | 82.90(7.23) / 83.30(1.91) / 97.67(0.73) / **97.99(0.73)** |
| | | Food-101 | 67.93(0.59) / 76.90(1.39) / 64.14(1.46) / **79.79(1.03)** | 89.87(0.44) / 92.06(0.71) / 80.38(3.83) / **93.98(0.54)** | 83.58(0.96) / 85.30(1.54) / 63.50(8.01) / **89.99(0.90)** |
| | | MNIST | 76.35(3.52) / 97.00(0.49) / **97.50(0.00)** / 95.15(1.51) | 94.54(1.02) / **99.69(0.20)** / 99.43(0.44) / 98.90(0.49) | 95.98(0.83) / **99.72(0.18)** / 99.68(0.25) / 99.06(0.42) |
| | | F-MNIST | 81.65(2.11) / **96.49(0.39)** / 92.79(4.59) / 94.83(0.71) | 95.76(0.50) / **99.51(0.13)** / 97.87(2.00) / 98.84(0.21) | 96.76(0.35) / **99.54(0.11)** / 97.95(2.26) / 98.94(0.17) |
| | | NotMNIST | 75.95(3.71) / 94.44(2.12) / **96.04(1.24)** / 95.75(1.09) | 93.72(1.49) / 98.61(0.95) / **99.18(0.60)** / 99.01(0.30) | 91.90(2.37) / 97.04(2.34) / **98.98(0.63)** / 98.62(0.45) |
| | | Gaussian | 63.72(14.12) / 97.48(0.03) / **97.50(0.00)** / **97.50(0.00)** | 91.96(4.29) / 99.30(0.47) / **100.00(0.00)** / **100.00(0.00)** | 95.17(2.27) / 99.54(0.26) / **100.00(0.00)** / **100.00(0.00)** |
| | | Uniform | 60.28(19.27) / 96.79(1.52) / **97.50(0.00)** / **97.50(0.00)** | 88.72(7.18) / 98.79(0.95) / **100.00(0.00)** / **100.00(0.00)** | 93.28(4.22) / 99.23(0.57) / **100.00(0.00)** / **100.00(0.00)** |
| | CIFAR-100 | TIN (c) | 64.67(2.28) / 84.84(4.02) / 81.53(4.16) / **91.71(0.85)** | 83.70(4.00) / 94.48(3.21) / 92.97(1.63) / **97.90(0.29)** | 84.98(6.13) / 94.16(4.45) / 93.00(1.86) / **98.03(0.24)** |
| | | TIN (c)* | 60.26(1.93) / 73.44(4.06) / 80.13(0.82) / **89.81(1.52)** | 79.32(4.14) / 88.54(4.27) / 93.18(0.39) / **97.31(0.45)** | 80.66(6.76) / 87.97(6.39) / 94.05(0.42) / **97.55(0.37)** |
| | | TIN (r) | 58.66(2.35) / 74.66(5.82) / 88.95(0.58) / **91.37(1.59)** | 77.07(6.35) / 88.14(6.92) / 96.81(0.27) / **97.82(0.53)** | 78.60(9.30) / 87.18(9.82) / 97.26(0.30) / **97.99(0.45)** |
| | | LSUN (c) | 62.53(0.41) / 84.21(1.09) / 77.94(5.23) / **88.48(0.77)** | 82.92(0.59) / 94.72(0.59) / 91.65(2.96) / **96.73(0.31)** | 85.41(0.77) / 94.80(0.69) / 91.67(2.98) / **96.89(0.28)** |
| | | LSUN (c)* | 58.61(0.57) / 73.30(1.74) / 70.53(0.95) / **85.26(0.79)** | 78.46(0.91) / 87.89(1.13) / 85.44(1.85) / **95.52(0.32)** | 80.86(0.93) / 87.55(1.35) / 86.39(2.20) / **95.82(0.29)** |
| | | LSUN (r) | 59.32(2.47) / 76.53(5.59) / 89.82(0.48) / **90.69(2.23)** | 78.44(5.41) / 90.38(4.76) / 97.00(0.15) / **97.59(0.75)** | 80.74(6.55) / 90.49(5.57) / 97.54(0.10) / **97.83(0.65)** |
| | | iSUN | 58.37(2.76) / 74.54(6.13) / 89.43(0.20) / **90.48(2.17)** | 76.89(6.28) / 88.27(6.49) / 97.04(0.10) / **97.45(0.73)** | 80.56(7.56) / 89.01(7.25) / 97.80(0.13) / **97.88(0.56)** |
| | | SVHN | 56.40(1.82) / 78.49(1.49) / 87.06(2.51) / **88.42(2.57)** | 77.36(2.83) / 91.60(0.73) / 96.48(0.68) / **96.90(0.79)** | 66.58(4.07) / 82.08(1.58) / **94.33(0.78)** / 93.95(1.33) |
| | | Food-101 | 62.64(0.63) / **70.96(1.24)** / 55.43(2.48) / 68.29(1.18) | 84.38(0.48) / **90.82(0.60)** / 67.14(1.39) / 90.79(0.49) | 77.21(0.77) / 85.64(1.04) / 46.59(4.01) / **86.96(0.59)** |
| | | STL-10 | 57.82(1.32) / 70.35(2.12) / **85.77(6.70)** / 75.20(3.96) | 76.53(2.23) / 85.41(2.55) / **91.64(5.03)** / 90.28(1.95) | 81.56(2.67) / 87.27(2.78) / 91.32(5.15) / **92.43(1.54)** |
| | | MNIST | 62.68(2.68) / 91.17(5.19) / **97.47(0.07)** / 86.62(5.51) | 81.56(3.83) / 97.57(2.07) / **99.81(0.19)** / 96.17(2.06) | 84.50(3.62) / 97.74(1.94) / **99.88(0.12)** / 96.62(1.82) |
| | | F-MNIST | 70.15(1.95) / 93.01(1.50) / **94.79(2.94)** / 94.69(1.10) | 88.52(1.04) / 98.27(0.49) / 98.58(1.11) / **98.92(0.35)** | 90.30(0.80) / 98.33(0.48) / 98.66(1.18) / **99.02(0.31)** |
| | | NotMNIST | 59.04(1.67) / 85.20(2.34) / **94.73(2.98)** / 84.44(3.09) | 79.97(1.37) / 94.73(1.37) / **98.69(1.02)** / 95.80(0.86) | 73.28(2.54) / 90.74(2.83) / **97.98(0.88)** / 93.87(1.12) |
| | | Gaussian | 47.54(0.05) / 77.72(19.57) / **97.50(0.00)** / **97.50(0.00)** | 53.73(13.50) / 93.44(6.60) / **100.00(0.00)** / 99.74(0.52) | 69.06(10.42) / 95.41(4.73) / **100.00(0.00)** / 99.84(0.30) |
| | | Uniform | 48.34(1.81) / 79.64(23.92) / **97.50(0.00)** / **97.50(0.00)** | 63.59(20.05) / 94.60(6.66) / **100.00(0.00)** / 99.71(0.56) | 75.90(13.16) / 96.41(4.48) / **100.00(0.00)** / 99.82(0.34) |
| WRN-28-10 | CIFAR-10 | TIN (c) | 78.71(1.31) / 88.38(1.69) / 86.46(2.18) / **93.26(0.96)** | 93.86(0.90) / 95.88(1.01) / 95.99(1.04) / **98.35(0.32)** | 94.45(1.23) / 95.38(1.23) / 96.55(0.98) / **98.53(0.27)** |
| | | TIN (c)* | 76.10(1.38) / 83.16(2.07) / **93.57(0.31)** / 92.63(1.26) | 91.79(1.57) / 92.17(2.19) / **98.50(0.11)** / 98.17(0.33) | 90.94(2.91) / 89.77(3.56) / **98.72(0.10)** / 98.42(0.27) |
| | | TIN (r) | 72.35(2.13) / 81.86(3.29) / **95.59(0.34)** / 91.05(2.24) | 89.21(2.65) / 90.60(3.21) / **99.15(0.18)** / 97.65(0.66) | 87.62(4.65) / 87.99(4.53) / **99.25(0.19)** / 97.94(0.54) |
| | | LSUN (c) | 82.07(1.93) / 90.24(0.84) / 78.82(2.25) / **95.58(0.21)** | 95.41(0.26) / 97.20(0.15) / 92.65(1.33) / **99.19(0.07)** | 96.19(0.18) / 97.28(0.17) / 93.12(1.38) / **99.23(0.07)** |
| | | LSUN (c)* | 78.31(2.05) / 85.89(1.53) / 88.26(1.26) / **94.96(0.16)** | 93.67(0.50) / 95.08(0.42) / 96.90(0.35) / **98.98(0.07)** | 93.90(0.56) / 94.42(0.56) / 97.46(0.27) / **99.06(0.05)** |
| | | LSUN (r) | 76.39(2.41) / 87.33(2.22) / **96.40(0.31)** / 94.02(1.29) | 92.45(1.48) / 94.48(1.70) / **99.37(0.13)** / 98.59(0.34) | 92.33(2.31) / 93.12(2.42) / **99.47(0.10)** / 98.79(0.28) |
| | | iSUN | 74.82(2.19) / 85.49(2.99) / **96.08(0.20)** / 93.47(1.35) | 91.22(2.05) / 93.25(2.43) / **99.29(0.10)** / 98.48(0.36) | 91.40(3.31) / 92.35(3.24) / **99.46(0.08)** / 98.82(0.26) |
| | | SVHN | 80.66(2.52) / 87.37(3.32) / 95.79(0.22) / **96.50(0.69)** | 94.43(1.30) / 93.34(3.60) / 99.28(0.09) / **99.52(0.24)** | 88.55(4.75) / 79.63(11.13) / 98.51(0.16) / **98.94(0.52)** |
| | | Food-101 | 71.61(1.04) / 75.92(1.90) / 71.73(1.64) / **78.73(1.42)** | 89.71(0.90) / 89.18(2.37) / 90.43(1.54) / **93.95(0.41)** | 76.04(3.29) / 71.88(6.62) / 84.07(3.15) / **90.41(0.47)** |
| | | MNIST | 72.35(5.94) / 92.32(2.53) / **97.42(0.05)** / 94.60(1.22) | 91.16(3.19) / 97.36(1.50) / **99.56(0.11)** / 98.60(0.39) | 91.37(3.95) / 97.03(1.79) / **99.71(0.07)** / 98.91(0.28) |
| | | F-MNIST | 81.20(1.34) / 92.25(1.25) / **96.24(0.68)** / 94.03(0.72) | 94.59(0.61) / 97.28(0.65) / **99.03(0.23)** / 98.60(0.24) | 94.91(0.79) / 96.65(0.91) / **99.28(0.21)** / 98.78(0.20) |
| | | NotMNIST | 81.27(2.34) / 94.93(0.51) / **97.33(0.14)** / 96.33(0.62) | 95.00(0.58) / 98.48(0.38) / **99.77(0.07)** / 99.11(0.28) | 92.37(1.43) / 96.20(1.28) / **99.65(0.10)** / 98.84(0.32) |
| | | Gaussian | 82.94(17.44) / 96.33(2.54) / **97.50(0.00)** / **97.50(0.00)** | 95.16(6.00) / 99.39(0.94) / **100.00(0.00)** / 99.98(0.02) | 96.47(4.70) / 99.48(0.82) / **100.00(0.00)** / 99.99(0.01) |
| | | Uniform | 78.59(14.28) / 96.65(1.53) / **97.50(0.00)** / **97.50(0.00)** | 95.30(2.90) / 99.41(0.53) / **100.00(0.00)** / 99.99(0.01) | 96.70(2.10) / 99.48(0.53) / **100.00(0.00)** / 99.99(0.01) |
| | CIFAR-100 | TIN (c) | 65.24(1.62) / 77.12(2.53) / 79.69(5.23) / **87.94(1.16)** | 84.47(1.24) / 91.72(1.10) / 92.58(2.60) / **96.76(0.34)** | 86.08(1.27) / 92.21(0.95) / 93.37(2.43) / **97.11(0.25)** |
| | | TIN (c)* | 61.96(0.78) / 71.32(1.31) / **87.88(0.94)** / 85.01(1.35) | 80.90(0.90) / 87.08(1.29) / **96.45(0.30)** / 95.91(0.42) | 81.38(1.79) / 86.72(1.71) / **97.02(0.29)** / 96.48(0.31) |
| | | TIN (r) | 58.89(0.99) / 71.53(2.26) / **91.88(0.30)** / 84.94(2.40) | 76.67(2.03) / 86.28(2.43) / **97.82(0.13)** / 95.84(0.67) | 76.74(3.62) / 85.82(3.03) / **98.09(0.10)** / 96.40(0.47) |
| | | LSUN (c) | 60.99(1.01) / 78.04(0.67) / 68.64(0.78) / **86.49(1.74)** | 81.91(1.31) / 91.75(0.44) / 80.48(1.14) / **96.09(0.62)** | 84.68(1.45) / 91.90(0.47) / 79.46(1.59) / **96.41(0.56)** |
| | | LSUN (c)* | 58.99(0.77) / 74.15(0.57) / 77.82(1.07) / **82.97(1.77)** | 79.17(1.25) / 88.06(0.46) / 91.13(0.52) / **94.92(0.65)** | 80.88(1.99) / 87.28(0.65) / 92.04(0.52) / **95.45(0.57)** |
| | | LSUN (r) | 58.98(1.47) / 72.74(2.41) / **91.98(0.32)** / 82.76(2.40) | 78.00(1.95) / 87.90(1.83) / **97.80(0.15)** / 95.18(0.86) | 79.25(2.28) / 87.83(1.99) / **98.12(0.13)** / 95.92(0.68) |
| | | iSUN | 58.54(1.43) / 71.72(2.57) / **91.69(0.46)** / 83.73(1.92) | 77.29(2.15) / 87.07(2.00) / **97.66(0.14)** / 95.39(0.55) | 80.27(2.59) / 88.13(1.99) / **98.18(0.10)** / 96.40(0.35) |
| | | SVHN | 58.13(2.90) / 83.04(1.69) / **92.39(1.42)** / 90.28(1.57) | 79.82(2.49) / 93.46(1.05) / **97.96(0.49)** / 97.52(0.41) | 66.44(4.78) / 84.81(3.06) / **96.05(0.68)** / 95.13(0.54) |
| | | Food-101 | 69.12(0.76) / 71.12(0.79) / 70.23(1.61) / **72.68(1.01)** | 89.25(0.40) / 90.76(0.35) / 91.15(0.66) / **92.53(0.38)** | 83.20(0.70) / 84.74(0.78) / 87.69(0.85) / **89.37(0.54)** |
| | | STL-10 | 58.23(0.55) / 66.14(0.39) / **86.69(2.54)** / 71.03(1.17) | 77.68(0.61) / 83.02(1.12) / **94.73(1.36)** / 88.89(0.51) | 81.66(1.01) / 85.27(1.49) / **95.52(1.95)** / 91.70(0.43) |
| | | MNIST | 61.79(5.62) / 87.27(4.78) / **94.15(3.68)** / 85.60(3.96) | 82.64(4.73) / 96.71(1.39) / **98.54(1.01)** / 96.03(1.49) | 85.99(3.87) / 97.18(1.17) / **98.89(0.65)** / 96.59(1.25) |
| | | F-MNIST | 72.39(2.13) / 88.50(1.70) / 87.63(0.67) / **92.48(1.22)** | 90.33(1.02) / 96.61(0.68) / 96.61(0.30) / **98.27(0.36)** | 91.74(0.87) / 96.71(0.75) / 97.36(0.24) / **98.46(0.31)** |
| | | NotMNIST | 56.15(1.61) / 81.37(2.36) / **96.48(0.38)** / 77.23(2.62) | 80.00(2.79) / 93.47(1.35) / **98.93(0.13)** / 93.05(1.47) | 73.99(4.65) / 89.37(2.60) / **98.84(0.15)** / 90.53(2.00) |
| | | Gaussian | 47.54(0.07) / 86.50(21.83) / **97.50(0.00)** / **97.50(0.00)** | 59.59(30.02) / 80.04(41.24) / **100.00(0.00)** / 99.82(0.15) | 73.02(23.16) / 85.54(30.09) / **100.00(0.00)** / 99.88(0.09) |
| | | Uniform | 47.51(0.02) / 75.44(20.61) / **97.50(0.00)** / **97.50(0.00)** | 60.88(14.02) / 92.66(8.46) / **100.00(0.00)** / 99.95(0.10) | 74.03(12.70) / 95.08(6.01) / **100.00(0.00)** / 99.97(0.07) |

tion. Note that iSUN is chosen as the validation OOD dataset for the hyperparameter determination of the leave-out ensemble, following [38].

## 2.11 Black Frame in the Cropped OOD Images

The OOD datasets, i.e., LSUN (cropped & resized), TinyImageNet (cropped & resized), and iSUN, provided in the authors' GitHub repo of ODIN [3] [6] are used in many studies [3, 27, 38]. As mentioned briefly in Sec. 2.4.1, we found that every image in the datasets of cropped images, i.e., Tiny ImageNet (cropped) and LSUN (cropped), unexpectedly has a black frame with two-pixel widths, as shown in Fig. 2.7. Those images is of $36 \times 36$ pixels (4 pixels larger than CIFAR images), implying that it is a mistake of the authors. In any case, adding a black frame is not invalid by itself, as any image could be an OOD sample. However, it will ease the problem without a doubt.

In our experiments, we used both of the corrected versions (indicated by *) and the original version. The results show that the proposed method achieves similar performance on both versions of the datasets. On the other hand, the other methods, ODIN [3], Mahalanobis detector [4] and Leave-out Ensemble [38], show more sensitive behaviors to the difference, as observable in Tables 2.4 - 2.9 in this chapter.

## 2.12 Equivalence Between Max-softmax With a High Temperature and Max-logit

In the earlier section, we mention that when a high temperature is used in the softmax function, using the max-softmax criteria for OOD detection is equivalent to using the max-logit criteria. A brief proof is given below.

When employing a temperature, the score (or posterior probability) $p_i$ of class $i \in [1, C]$ is given by

$$p_i = \mathrm{softmax}_i(x/t) = \frac{e^{x_i/t}}{\sum_{i=1}^{C} e^{x_j/t}},$$

[6]https://github.com/facebookresearch/odin

Table 2.8: Performance of OOD detection by ensemble models (five networks) in the one-vs-one evaluation (DenseNet).

| Network | ID | OOD | Accuracy at TPR= 95% | AUROC | AUPR-In |
|---|---|---|---|---|---|
| | | | Leave-out [38] / Leave-out* / Ours | | |
| Dense-100-12 | CIFAR-10 | TinyIm (c) | 96.89 / 96.76 / 96.43 | 99.65 / 99.66 / 99.43 | 99.68 / 99.67 / 99.48 |
| | | TinyIm (c)* | - / 94.83 / 96.21 | - / 98.98 / 99.33 | - / 99.05 / 99.39 |
| | | TinyIm (r) | 96.04 / 96.21 / 96.14 | 99.34 / 99.45 / 99.36 | 99.37 / 99.48 / 99.40 |
| | | LSUN (c) | 95.79 / 95.65 / 96.51 | 99.25 / 99.27 / 99.51 | 99.29 / 99.32 / 99.53 |
| | | LSUN (c)* | - / 91.04 / 96.06 | - / 97.83 / 99.33 | - / 98.00 / 99.36 |
| | | LSUN (r) | 97.12 / 96.71 / 96.79 | 99.75 / 99.67 / 99.61 | 99.77 / 99.68 / 99.64 |
| | | iSUN | - / 96.47 / 96.69 | - / 99.60 / 99.61 | - / 99.62 / 99.67 |
| | | SVHN | - / 81.09 / 96.53 | - / 94.39 / 99.54 | - / 95.06 / 98.93 |
| | | Food-101 | - / 75.64 / 85.06 | - / 92.85 / 95.89 | - / 94.13 / 93.25 |
| | | MNIST | - / 97.19 / 97.01 | - / 99.76 / 99.53 | - / 99.79 / 99.61 |
| | | F-MNIST | - / 96.59 / 96.60 | - / 99.58 / 99.45 | - / 99.62 / 99.51 |
| | | NotMNIST | - / 93.39 / 97.11 | - / 98.64 / 99.45 | - / 98.83 / 99.24 |
| | | Gausian | 96.20 / 97.50 / 97.50 | 98.55 / 99.99 / 100.00 | 98.94 / 99.99 / 100.00 |
| | | Uniform | 97.50 / 97.50 / 97.50 | 99.84 / 99.96 / 100.00 | 99.89 / 99.97 / 100.00 |
| | CIFAR-100 | TinyIm (c) | 93.36 / 95.11 / 94.44 | 98.43 / 99.00 / 98.78 | 98.58 / 99.05 / 98.86 |
| | | TinyIm (c)* | - / 88.78 / 93.09 | - / 96.79 / 98.30 | - / 97.03 / 98.46 |
| | | TinyIm (r) | 87.24 / 91.45 / 93.89 | 96.27 / 97.80 / 98.60 | 96.66 / 98.01 / 98.72 |
| | | LSUN (c) | 90.16 / 89.34 / 91.54 | 97.37 / 97.05 / 97.80 | 97.62 / 97.26 / 97.91 |
| | | LSUN (c)* | - / 75.23 / 88.55 | - / 90.75 / 96.74 | - / 91.27 / 96.99 |
| | | LSUN (r) | 89.39 / 93.25 / 93.16 | 97.03 / 98.37 / 98.38 | 97.37 / 98.52 / 98.55 |
| | | iSUN | - / 90.91 / 92.93 | - / 97.53 / 98.23 | - / 97.66 / 98.53 |
| | | SVHN | - / 50.84 / 92.82 | - / 75.79 / 98.22 | - / 81.16 / 96.39 |
| | | Food-101 | - / 74.44 / 76.42 | - / 92.39 / 93.76 | - / 93.82 / 91.23 |
| | | STL-10 | - / 86.56 / 78.48 | - / 96.34 / 92.04 | - / 96.78 / 93.93 |
| | | MNIST | - / 96.35 / 89.23 | - / 99.27 / 97.35 | - / 99.39 / 97.68 |
| | | F-MNIST | - / 96.73 / 96.88 | - / 99.66 / 99.64 | - / 99.67 / 99.67 |
| | | NotMNIST | - / 92.20 / 87.92 | - / 98.07 / 96.97 | - / 98.25 / 95.57 |
| | | Gausian | 57.64 / 81.18 / 97.50 | 92.00 / 95.57 / 100.00 | 94.77 / 97.36 / 100.00 |
| | | Uniform | 78.24 / 95.72 / 97.50 | 94.89 / 97.73 / 100.00 | 96.36 / 98.61 / 100.00 |

Table 2.9: Performance of OOD detection by ensemble models (five networks) in the one-vs-one evaluation (Wide Resnet).

| Network | ID | OOD | Accuracy at TPR= 95% | AUROC | AUPR-In |
|---|---|---|---|---|---|
| | | | Leave-out [38] / Leave-out* / Ours | | |
| WRN-28-10 | CIFAR-10 | TinyIm (c) | 97.09 / 96.35 / 94.83 | 99.75 / 99.54 / 98.93 | 99.77 / 99.57 / 99.05 |
| | | TinyIm (c)* | - / 93.47 / 94.62 | - / 98.56 / 98.77 | - / 98.65 / 98.95 |
| | | TinyIm (r) | 96.03 / 95.09 / 93.41 | 99.36 / 99.10 / 98.41 | 99.40 / 99.13 / 98.62 |
| | | LSUN (c) | 96.54 / 94.97 / 96.42 | 99.55 / 99.09 / 99.46 | 99.57 / 99.16 / 99.49 |
| | | LSUN (c)* | - / 88.93 / 95.91 | - / 97.14 / 99.30 | - / 97.33 / 99.35 |
| | | LSUN (r) | 97.06 / 95.88 / 95.74 | 99.70 / 99.39 / 99.14 | 99.72 / 99.37 / 99.27 |
| | | iSUN | - / 95.33 / 95.53 | - / 99.22 / 99.06 | - / 99.22 / 99.28 |
| | | SVHN | - / 73.78 / 96.99 | - / 91.80 / 99.73 | - / 93.15 / 99.36 |
| | | Food-101 | - / 66.64 / 81.60 | - / 87.37 / 95.10 | - / 89.23 / 92.34 |
| | | MNIST | - / 95.52 / 96.12 | - / 99.09 / 98.98 | - / 99.22 / 99.22 |
| | | F-MNIST | - / 96.59 / 95.33 | - / 99.59 / 99.04 | - / 99.60 / 99.16 |
| | | NotMNIST | - / 92.33 / 96.99 | - / 97.91 / 99.38 | - / 98.05 / 99.19 |
| | | Gausian | 89.31 / 97.50 / 97.50 | 96.77 / 99.97 / 100.00 | 97.78 / 99.98 / 100.00 |
| | | Uniform | 97.50 / 97.50 / 97.50 | 99.58 / 99.98 / 100.00 | 99.71 / 99.98 / 100.00 |
| | CIFAR-100 | TinyIm (c) | 92.92 / 92.88 / 90.58 | 98.22 / 98.33 / 97.62 | 98.39 / 98.46 / 97.89 |
| | | TinyIm (c)* | - / 84.19 / 87.83 | - / 95.34 / 96.80 | - / 95.69 / 97.26 |
| | | TinyIm (r) | 85.24 / 88.74 / 87.09 | 95.18 / 96.89 / 96.64 | 95.50 / 97.16 / 97.11 |
| | | LSUN (c) | 90.39 / 83.34 / 88.61 | 97.38 / 95.43 / 97.00 | 97.62 / 96.03 / 97.26 |
| | | LSUN (c)* | - / 68.83 / 85.23 | - / 87.66 / 95.87 | - / 88.55 / 96.33 |
| | | LSUN (r) | 89.24 / 92.17 / 84.58 | 96.77 / 97.98 / 95.97 | 97.03 / 98.13 / 96.65 |
| | | iSUN | - / 90.33 / 85.66 | - / 97.31 / 96.17 | - / 97.41 / 97.05 |
| | | SVHN | - / 51.65 / 92.66 | - / 76.30 / 98.20 | - / 80.87 / 96.41 |
| | | Food-101 | - / 68.62 / 78.00 | - / 89.87 / 94.38 | - / 91.49 / 92.02 |
| | | STL-10 | - / 80.86 / 72.86 | - / 94.33 / 90.16 | - / 95.00 / 92.72 |
| | | MNIST | - / 93.84 / 86.94 | - / 98.62 / 96.79 | - / 98.82 / 97.23 |
| | | F-MNIST | - / 97.05 / 95.37 | - / 99.77 / 99.16 | - / 99.79 / 99.24 |
| | | NotMNIST | - / 86.31 / 78.95 | - / 96.53 / 94.32 | - / 97.03 / 92.40 |
| | | Gausian | 47.55 / 97.50 / 97.50 | 83.44 / 99.65 / 99.98 | 89.43 / 99.79 / 99.99 |
| | | Uniform | 48.37 / 97.50 / 97.50 | 93.04 / 99.60 / 100.00 | 88.64 / 99.76 / 100.00 |

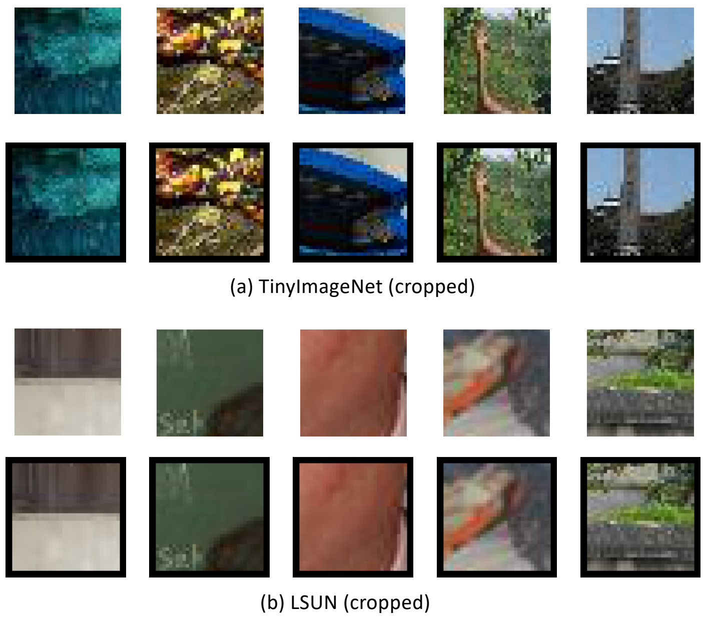


Figure 2.7: The images with and without a black frame. (a) TinyImageNet (cropped). (b) LSUN (cropped).

where $x_i$ is the logit of class $i$ and $t$ is the temperature. As shown in Hinton et al. [37], when $t$ is large, $p_i$ can be approximated as

$$p_i \approx \frac{1 + x_i/t}{N}.$$

The ODIN [3] employs the max-softmax criteria with a high temperature for OOD detection, which is written using the above approximation as

$$\frac{1 + x_{\max}/t}{N} < \tau,$$

where $x_{\max}$ is the maximum of the logits. This is rewritten as

$$x_{\max} < t(N\tau - 1).$$

By regarding the rhs as a new threshold, this coincides with the max-logit criteria.

Table 2.10: Comparison of computational time (per batch of 128 samples) of the three methods.

| Network | Time (second) | | |
|---|---|---|---|
| | Mahalanobis | ODIN | Cosine |
| Dense-100-12 | 0.67 | 0.19 | **0.08** |
| WRN-28-10 | 1.61 | 0.61 | **0.22** |

## 2.13 Computational Cost

While the proposed method needs only the standard forward propagation to perform OOD detection, the previous methods, particularly those showing good performance in the one-vs-one evaluation, employ a lot more complicated computation, such as input perturbation [3, 4]. We measure the computational time that ODIN, the Mahalanobis detector, and ours need to get the results. Table 2.10 shows the average time per batch containing 128 samples.

## 2.14 Summary and Conclusions

In this chapter, we have presented a novel method for OOD detection, and experimentally confirmed its superiority to existing approaches. We started our discussion with the observation that existing methods have hyperparameters specific to OOD detection, and their performance can be sensitive to their determination. The proposed method does not have such hyperparameters. It is based on the softmax of scaled cosine similarity and can be used with any networks by replacing their output layer. Training is performed by the standard method, i.e., minimizing a cross-entropy loss on the target classification task. Although a similar approach has already been employed in metric learning methods, the proposed method has several technical differences, which are essential to achieve high OOD detection performance, as was demonstrated in our ablation test. We have shown experimental comparisons between the proposed method and the existing methods using two different evaluation methods, i.e., the less-biased evaluation recently proposed in [1] and the conventional one-vs-one evaluation. In the former evaluation, which takes the above issue with hyperparameter determination into account, the proposed method shows clear superiority to others. Our method also shows at least comparable performance to them in the conventional evaluation. These results support the practicality of the proposed method in real-world applications. Lastly, we have briefly discussed why cosine similarity is effective for OOD detection.

# Chapter 3

# Practical Evaluation of Out-of-Distribution Detection Methods for Image Classification

## 3.1 Introduction

Despite their high performance on various visual recognition tasks, convolutional neural networks (CNNs) often show unpredictable behaviors against out-of-distribution (OOD) inputs, i.e., those sampled from a different distribution from the training data. For instance, CNNs often classify irrelevant images to one of the known classes with high confidence. A visual recognition system should desirably be equipped with an ability to detect such OOD inputs upon its real-world deployment.

There are many studies of OOD detection that are based on diverse motivations and purposes. However, as far as the recent studies targeted at visual recognition are concerned, most of them follow the work of [2], which provides a formal problem statement of OOD detection and an experimental procedure to evaluate the performance of methods. Employing this procedure, the recent studies focus mainly on increasing detection accuracy, where the performance is measured using the same datasets.

On the one hand, the employment of the experimental procedure has arguably bought about the rapid progress of research in a short period. On the other hand, little attention has been paid to how well the employed procedure models real-world problems and applications. They are diverse in purposes and domains, which obviously cannot be covered by the single problem setting with a narrow range of datasets.

In this study, to address this issue, we consider multiple, more realistic scenarios of the application of OOD detection, and then experimentally compare the representative methods. To be specific, we consider the three scenarios: *detection of irrelevant inputs*, *detection of novel class inputs*, and *detection of domain shift*. The first two scenarios differ in the closeness between ID samples and OOD samples.

Unlike the first two, domain shift detection is not precisely OOD detection. Nonetheless, it is the same as the other two in that what we want is to judge if the model can make a meaningful inference for a novel input. In other words, we can generalize OOD detection to the problem of judging this. Then, the above three scenarios are naturally fallen into the same group of problems, and it becomes natural to consider applying OOD detection methods to the third scenario. It is noteworthy that domain shift detection has been poorly studied in the community. Despite many demands from practitioners, there is no established method in the context of deep learning for image classification. Based on the above generalization of OOD detection, we propose a meta-approach in which any OOD detection method can be used as its component.

For each of these three scenarios, we compare the following methods: the confidence-based baseline [2], MC dropout [13], ODIN [3], cosine similarity [5, 74], and the Mahalanobis detector [4]. Domain shift detection is studied in [75] with natural language processing tasks, where proxy-A distance (PAD) is reported to perform the best; thus we test it in our experiments.

As for choosing the compared methods, we follow the argument shared by many recent studies [1, 5, 26, 27, 74] that OOD detection methods should not assume the availability of explicit OOD samples at training time. Although this may sound obvious considering the nature of OOD, some of the recent methods (e.g., [3, 4]) use a certain amount of OOD samples as *validation* data to determine their hyperparameters. The recent studies, [1, 74], show that these methods do perform poorly when encountering OOD inputs sampled from a different distribution from the assumed one at test time. Thus, for ODIN and the Mahalanobis detector, we employ their variants [4, 5] that can work without OOD samples. The other compared methods do not need OOD samples.

The contribution of this study are summarized as follows. i) Listing three problems that practitioners frequently encounter, we evaluate the existing OOD detection methods on each of them. ii) We show a practical approach to domain shift detection that is applicable to CNNs for image classification. iii) We show experimental evaluation of representative OOD detection methods on these problems, revealing each method's effectiveness and ineffectiveness in each scenario.

## 3.2 Problems and Methods

### 3.2.1 Practical Scenarios of OOD Detection

We consider image recognition tasks in which a CNN classifies a single image $x$ into one of $C$ known classes. The CNN is trained using pairs of $x$ and its label, and $x$ is sampled according to $x \sim p(x)$. At test time, it will encounter an unseen input $x$, which is usually from $p(x)$ but is sometimes from $p'(x)$, a different, unknown distribution. In this study, we consider the following three scenarios.

**Detecting Irrelevant Inputs** The new input $x$ does not belong to any of the known classes and is out of concern. Suppose we want to build a smartphone app that recognizes dog breeds. We train a CNN on a dataset containing various dog images, enabling it to perform the task with reasonable accuracy. We then point the smartphone to a sofa and shoot its image, feeding it to our classifier. It could classify the image as a *Bull Terrier* with high confidence. Naturally, we want to avoid this by detecting the irrelevance of $x$. Most studies of OOD detection assumes this scenario for evaluation.

**Detecting Novel Classes** The input $x$ belongs to a novel class, which differs from any of $C$ known classes, and furthermore, we want our CNN to learn to classify it later, e.g., after additional training. For instance, suppose we are building a system that recognizes insects in the wild, with an ambition to make it cover all the insects on the earth. Further, suppose an image of one of the endangered (and thus rare) insects is inputted to the system while operating it. If we can detect it as a novel class, we would be able to update the system in several ways. The problem is the same as the first scenario in that we want to detect whether $x \sim p(x)$ or not. The difference is that $x$ is more similar to samples of the learned classes, or equivalently, $p'(x)$ is more close to $p(x)$, arguably making the detection more difficult. Note that in this study, we don't consider distinguishing whether $x$ is an irrelevant input or a novel class input, for the sake of simplicity. We left it for a future study.

**Detecting Domain Shift** The input $x$ belongs to one of $C$ known classes, but its underlying distribution is $p'(x)$, not $p(x)$. We are especially interested in the case where a distributional shift $p(x) \rightarrow p'(x)$ occurs either suddenly or gradually while running a system for the long term. Our CNN may or may not generalize beyond this shift to $p'(x)$. Thus, we want to detect if it does not. If we can do this, we would take some actions, such as re-training the network with new training data [75]. We consider the case where no information is available other than the incoming inputs $x's$.

A good example is a surveillance system using a camera deployed outdoor. Let us assume the images' quality deteriorates after some time since its deployment, for instance, due to the camera's aging. Then, the latest images will follow a different distribution from that of the training data. Unlike the above two cases where we have to decide for a single input, we can use multiple inputs; we should, especially when the quality of input images deteriorate gradually as time goes.

The problem here has three differences from the above two scenarios. First, the input is a valid sample belonging to a known class, neither an irrelevant sample nor a novel class sample. Second, we are basically interested in the accuracy of our CNN with the latest input(s) and not in whether $x \sim p(x)$ or $p'(x)$. Third, as mentioned above, we can use multiple inputs $\{x_i\}_{i=1,\dots,n}$ for the judgment.

Additional remarks on this scenario. Assuming a temporal sequence of inputs, the distributional shift is also called *concept drift* [76]. It includes several different subproblems, and the one considered here is called *virtual* concept drift in its terminology. Mathematically, concept drift occurs when $p(x, y)$ changes with time. It is called *virtual* when $p(x)$ changes while $p(y|x)$ does not change. Intuitively, this is the case where the classes (i.e., concept) remain the same but $p(x)$ changes, demanding the classifier to deal with inputs drawn from $p'(x)$. Then, we are usually interested in predicting if $x$ lies in a region of the data space for which our classifier is well trained and can correctly classify it. If not, we might want to retrain our classifier using additional data or invoke unsupervised domain adaptation methods [77, 78].

### 3.2.2 Compared Methods

We select five representative OOD detection methods that do not use real OOD samples to be encountered at test time.

**Baseline: Max-softmax** [2] showed that the maximum of the softmax outputs, or confidence, can be used to detect OOD inputs. We use it as the score of an input being in-distribution (ID). We will refer to this method as *Baseline*. It is well known that the confidence can be calibrated using temperature to better represent classification accuracy [11, 12]. We also evaluate this calibrated confidence, which will be referred to as *Calib*.

**MC Dropout** The confidence (i.e., the max-softmax) is also thought of as a measure of uncertainty of prediction, but it captures only aleatoric uncertainty [28]. Bayesian neural networks (BNNs) can also take epistemic uncertainty into account, which is theoretically more relevant to OOD detection. MC (Monte-Carlo) dropout [13] is an approximation of BNNs that is computationally more efficient than an ensemble of networks [17]. To be specific, using dropout [79] at test time provides multiple prediction samples, from which the average of their max-softmax values is calculated and used as ID score.

**Cosine Similarity** It is recently shown in [5, 74] that using scaled cosine similarities at the last layer of a CNN, similar to the angular softmax for metric learning, enables accurate OOD detection. To be specific, the method first computes cosine similarities between the feature vector of the final layer and class centers (or equivalently, normalized weight vectors for classes). They are multiplied with a scale and then normalized by softmax to obtain class scores. The scale, which is the inverse temperature, is predicted from the same feature vector. These computations are performed by a single layer replacing the last layer of a standard CNN. The maximum of the cosine similarities (without the scale) gives ID score. The method is free of hyperparameters for OOD detection. We will refer to it as *Cosine*.

**ODIN (with OOD-sample Free Extension)** ODIN was proposed by [3] to improve *Baseline* by perturbing an input $x \to x + \epsilon \cdot \mathrm{sgn}(\delta x)$ in the direction $\delta x$ of maximally increasing the max-softmax and also by temperature scaling. Thus, there are two hyperparameters, the perturbation size $\epsilon$ and the temperature $T$. In [3], they are chosen by assuming the availability of explicit OOD samples. Recently, [5] proposed to select $\epsilon \leftarrow \mathrm{argmax}_{\epsilon} \sum y_{\kappa}(x+\epsilon\cdot\mathrm{sgn}(\delta x))$, where $y_{\kappa}$ is the max-softmax and the summation is taken over ID samples in the validation set. As for the temperature, they set $T = 1000$. ID score is given by $y_{\kappa}(x + \epsilon \cdot \mathrm{sgn}(\delta x))$. To distinguish from the original ODIN, we refer to this as ODIN*.

**Mahalanobis Detector** The above three methods are based on the confidence. Another approach is to formulate the problem as unsupervised anomaly detection. [4] proposed to model the distribution of intermediate layer's activation by a Gaussian distribution for each class but with a shared covariance matrix among the classes. Given an input, the Mahalanobis distance concerning the predicted class is calculated at each layer. A score for OOD is given by the weighted sum of those calculated at different layers. The weights are predicted by logistic regression, which is determined by assuming the availability of OOD samples. To be free from the assumption, another method is suggested that generates adversarial examples from ID samples and regard them as OOD samples. It is also reported in [5] that setting all the weights to one works reasonably well. We evaluate the last two methods that do not need OOD samples. Although the original method optionally uses input perturbation similar to ODIN, we do not use it because our experiments show that its improvement is very small despite its high computational cost.

**Effects of Fine-tuning a Pre-trained Network** It has been well known that fine-tuning a pre-trained network on a downstream task improves its prediction accuracy, especially when a small amount of training data is available. It was pointed out in [80] that the improvement is little when there is sufficient training data. [56] then show that even in that case, using a pre-trained network helps increase the overall robustness of the inference. It includes improved OOD detection performance, in addition to robustness to adversarial attacks, better calibration of confidence, robustness to covariate shift. However, their experimental validation is performed only on a single configuration with a few datasets. It remains unclear if the improvement can generalize to a broader range of purposes and settings that may differ in image size, the number of training samples, and ID/OOD combinations.

## 3.3 Experimental Results

We use Resnet-50 [81] for a base network. We use it as is for *Baseline*, *ODIN**, and *Mahalanobis*, which share the same networks with the same weights, which will be referred to as *Standard*. We apply dropout to the last fully-connected layer with $p = 0.5$ and draw ten samples for *MC dropout*. We modify the last layer and the loss function for *Cosine*, following [74]. We use the ImageNet pre-trained model provided by the Torchvision libraryfor their pre-trained models. We employ AUROC to evaluate OOD detection performance with the first two scenarios, following previous studies.

### 3.3.1 Detection of Irrelevant Inputs

We employ the following five tasks and datasets: dog breed recognition (120 classes and 10,222 images; [82]), plant seeding classification (12 classes and 5,544 images; [83]), Food-101 (101 classes and 101,000 images; [67]), CUB-200 (200 classes and 11,788 images; [84]), and Stanford Cars (196 classes and 16,185 images; [85]). These datasets will be referred to as *Dog*, *Plant*, *Food*, *Bird*, and *Cars*. They are diverse in terms of image contents, the number of classes, difficulty of tasks (e.g., fine-grained/coarse-grained), etc. Choosing one of the five as ID and training a network on it, we regard each of the other four as OOD, measuring the OOD detection performance of each method on the $5 \times 4$ ID-OOD combination. We train each network for three times to measure the average and standard deviation for each configuration. Table 3.1 shows the accuracy of the five datasets/tasks for the three networks (i.e., *Standard*, *MC dropout*, and *Cosine*) trained from scratch and fine-tuned from a pre-trained model, respectively. It is seen that there is large gap between training-from-scratch and fine-tuning a pre-trained model for the datasets with fewer training samples.

Figure 3.1: Example images for the five datasets.

Table 3.1: Classification accuracy (mean and standard deviation in parenthesis) of the three networks on the five datasets/tasks.

| Dataset | Train-from-scratch | | | Fine-tuning | | |
|---|---|---|---|---|---|---|
| | *Standard* | *MC dropout* | *Cosine* | *Standard* | *MC dropout* | *Cosine* |
| *Dog* | 26.7(3.4) | 28.9(2.8) | 36.2(1.4) | 79.4(0.1) | 79.3(0.3) | 78.5(0.3) |
| *Plant* | 94.1(0.4) | 94.7(0.2) | 95.8(0.9) | 95.2(0.6) | 95.5(0.5) | 92.7(2.6) |
| *Food* | 75.5(1.0) | 76.4(0.2) | 76.6(0.1) | 80.5(0.0) | 80.7(0.1) | 79.2(0.1) |
| *Bird* | 24.7(0.9) | 28.5(0.6) | 31.3(2.4) | 71.9(0.3) | 72.4(0.4) | 70.1(0.3) |
| *Car* | 18.2(3.8) | 22.0(1.6) | 36.0(6.2) | 77.6(0.3) | 77.7(0.3) | 73.7(0.6) |

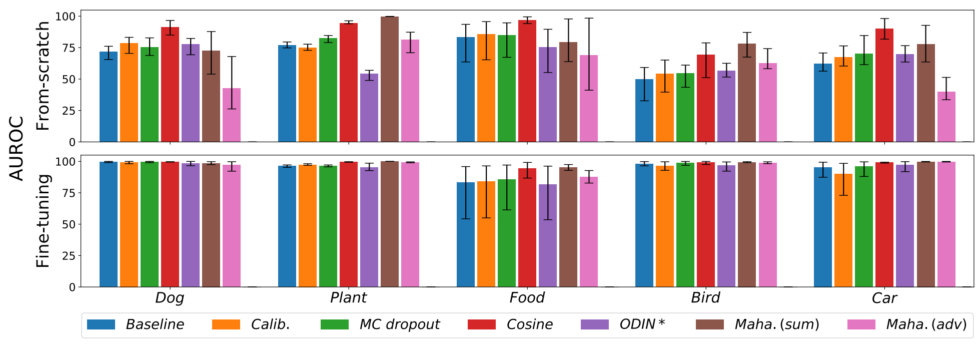


Figure 3.2: OOD detection performance of the compared methods. '*Dog*' indicates their performance when *Dog* is ID and all the other four datasets are OOD, etc. Each bar shows the average AUROC of a method, and the error bar indicates its minimum and maximum values. Upper: The networks are trained from scratch. Lower: Pre-trained models are fine-tuned.

Figure 3.2 shows the average AUROC of the compared OOD detection methods for each ID dataset over the four OOD datasets and three trials for each. The error bars indicate the minimum and maximum of AUROC. The full results for each of the twenty ID-OOD pairs are reported in Tables 3.5 and 3.6 in Sec. 3.4. The upper row of Fig. 3.2 shows the results with the networks trained from scratch. It is seen that the ranking of the compared methods are mostly similar for different ID datasets. For the five datasets, *Cosine* is consistently among the top group; *Mahalanobis* will be ranked next, since it performs mediocre for *Dog* and *Food*. For the tasks with low classification accuracy, *Dog*, *Bird*, and *Car*, as shown in Table 3.1, the OOD detection accuracy tends to be also low; however, there is no tendency in the ranking of the OOD detection methods depending on the ID classification accuracy.

The lower row of Fig. 3.2 shows the results with the fine-tuned networks. It is first observed for any dataset and method that the OOD detection accuracy is significantly higher than the networks trained from scratch. This reinforces the argument made by [56] that the use of pre-trained networks improves OOD detection performance. Furthermore, the performance increase is a lot higher for several cases than reported in their experiments that use CIFAR-10/100 and Tiny ImageNet [64]. The detection accuracy is pushed to a near-maximum for each case. Thus, there is only a little difference among the methods; *Cosine* and *Mahalanobis*(sum) shows slightly better performance for some datasets.

### 3.3.2 Detection of Novel Classes

We conducted two experiments with different datasets. The first experiment uses the Oxford-IIIT Pet dataset [86], consisting of 25 dog breeds and 12 cat breeds. We use only the dog breeds and split them into 20 and 5 breeds. We then train each network on the first 20 dog breeds using the standard train/test splits per class. The remaining five breeds (i.e., *Scottish Terrier, Shiba Inu, Staffordshire Bull Terrier, Wheaten Terrier, Yorkshire Terrier*) are treated as OOD. It should be noted that the ImageNet dataset contains 118 dog breeds, some of which overlap with them. We intentionally leave this overlap to simulate a similar situation that could occur in practice. In the second experiment, we use the Food-101 dataset. We remove eight classes[1] contained in the ImageNet dataset. We split the remaining 93 classes into 46 and 47 classes, called *Food-A* and *-B*, respectively. Each network is trained on *Food-A*. We split *Food-A* into 800/100/100 samples per class to form train/val/test sets. Treating *Food-B* as OOD, we evaluate the methods' performance.

Table 3.2 shows the methods' performance of detecting OOD samples (i.e., novel samples). In the table we separate the Mahalanobis detector and the others; the latter are all based on confidence or its variant, whereas Mahalanobis is not. The ranking of the methods is similar between the two experiments. *Cosine* attains the top performance for both of the two training methods. While this is similar to the results of irrelevant sample detection (Fig. 3.2), the gap to the second best group

[1]Apple Pie, Breakfast Burrito, Chocolate Mousse, Gaucamole, Hamburger, Hot Dog, Ice Cream, Pizza

Table 3.2: Novel class detection performance of the compared methods measured by AUROC.

| Method | *Dog* | | *Food-A* | |
|---|---|---|---|---|
| | From-scratch | Fine-tuning | From-scratch | Fine-tuning |
| *Baseline* | 61.1(0.8) | 88.7(1.0) | 82.5(0.1) | 84.6(0.2) |
| *Calib.* | 62.9(0.5) | 86.5(1.1) | 83.4(0.1) | 84.8(0.2) |
| *MC dropout* | 61.7(0.4) | 89.8(0.8) | 82.7(0.1) | 84.7(0.1) |
| *Cosine* | **68.8(1.3)** | **94.1(0.8)** | **83.7(0.1)** | **85.7(0.3)** |
| *ODIN** | 59.9(0.5) | 85.3(1.4) | 77.4(0.1) | 74.7(0.3) |
| *Maha. (sum)* | 52.3(1.2) | 78.7(1.4) | 51.3(0.1) | 61.9(0.5) |
| *Maha. (adv)* | 49.0(0.2) | 65.8(1.5) | 51.8(0.3) | 57.1(7.2) |

(*Baseline*, *Calib.*, and *MC dropout*) is much larger here; this is significant for training from scratch. Another difference is that neither variant of *Mahalanobis* performs well; they are even worse than *Baseline*. This will be attributable to the similarity between ID and OOD samples here. The classification accuracy of the original tasks, *Dog* and *Food-A* are given in Table 3.7 in Sec. 3.5.

### 3.3.3 Detection of Domain Shift

#### Problem Formulation

Given a network trained on a dataset $\mathcal{D}_s$, we wish to estimate its classification error on a different dataset $\mathcal{D}_t$. In practice, a meta-system monitoring the network estimates the classification error on each of the incoming datasets $\mathcal{D}_t^{(1)}, \mathcal{D}_t^{(2)}, \cdots$, which are chosen from the incoming data stream. It issues an alert if the predicted error for the latest $\mathcal{D}_t^{(T)}$ is higher than the pre-fixed target.

We use an OOD score $S$ for this purpose. To be specific, given $\mathcal{D}_t = \{x_i\}_{i=1,\dots,n}$, we calculate an average of the score $\overline{S} = \sum_i^n S_i/n$, where $S_i$ is the OOD score for $x_i$; note that an OOD score is simply given by a negative ID score. We want to use $\overline{S}$ to predict the classification error $\overline{\text{err}} = \sum_{i=1}^n 1(y_i = t_i)/n$, where $y$ and $t$ are a prediction and the true label, respectively. Following [75], we train a regressor $f$ to do this, as $\overline{\text{err}} \sim f(\overline{S})$. We assume multiple labeled datasets $\mathcal{D}_o$'s are available, each of which do not share inputs with $\mathcal{D}_s$ or $\mathcal{D}_t$. Choosing a two-layer MLP for $f$, we train it on

Table 3.3: Errors of the predicted classification error by the compared methods.

| Method | *Food-A* (From-scratch) | | *Food-A* (Fine-tuning) | | ImageNet | |
|---|---|---|---|---|---|---|
| | MAE | RMSE | MAE | RMSE | MAE | RMSE |
| *Baseline* | 15.8(3.0) | 20.5(3.7) | 6.4(1.3) | 7.9(1.6) | 4.6(0.8) | 6.3(1.0) |
| *Calib.* | 15.0(2.9) | 19.6(3.4) | 6.3(1.3) | 7.9(1.6) | 4.3(0.8) | 6.0(1.0) |
| *MC dropout* | 15.3(2.7) | 19.7(3.0) | **5.8(1.1)** | **7.2(1.4)** | 4.0(0.7) | 5.3(0.9) |
| *Cosine* | **6.6(1.3)** | **8.2(1.6)** | 6.1(1.6) | 7.5(2.2) | **3.8(0.9)** | **4.7(1.1)** |
| *ODIN** | 14.7(1.9) | 17.4(2.2) | 8.9(1.3) | 10.8(1.4) | 9.1(0.8) | 12.3(1.2) |
| *Maha. (sum)* | 15.3(1.5) | 18.4(1.8) | 15.6(1.5) | 18.9(2.1) | 15.1(2.2) | 18.5(2.9) |
| *Maha. (adv)* | 14.3(1.5) | 17.5(2.0) | 19.1(15.8) | 24.1(27.6) | 16.1(1.7) | 19.6(2.6) |
| *PAD* | 16.3(1.5) | 19.2(1.9) | 17.5(1.3) | 20.5(1.6) | 11.0(1.1) | 12.9(1.2) |

$\mathcal{D}_o$'s plus $\mathcal{D}_s$. As they have labels, we can get the pair of $\overline{\mathrm{err}}$ and $\overline{S}$ for each of them. Note that $\mathcal{D}_t$ does not have labels.

It is reported in [75] that Proxy-A Distance (PAD) [87] performs well on several NLP tasks. Thus, we also test this method (rigorously, the one called PAD$^*$ in their paper) for comparisons. It first trains a binary classifier using portions of $\mathcal{D}_s$ and $\mathcal{D}_t$ to distinguish the two. Then, the classifier's accuracy is evaluated on the held-out samples of $\mathcal{D}_s$ and $\mathcal{D}_t$, which is used as a metric of the distance between their underlying distributions. Intuitively, the classification is easy when their distance is large, and vice versa. We train $f$ using $1 -$ (mean absolute error) for $S$ as in the previous work.

### Domain Shift by Image Corruption

We first consider the case when the shift is caused by the deterioration of image quality. An example is a surveillance camera deployed in an outdoor environment. Its images are initially of high quality, but later their quality deteriorates gradually or suddenly due to some reason, e.g., dirt on the lens, failure of focus adjustment, seasonal/climate changes, etc. We want to detect it *if it affects classification accuracy.* To simulate multiple types of image deterioration, we employ the method and code for generating image corruption developed by [6]. It can generate 19 types of image corruptions, each of which has five levels of severity.

We consider two classification datasets/tasks, *Food-A* (i.e., 46 selected classes from Food-101 as explained in Sec. 3.3.2) and ImageNet (the original 1,000 object classification). For *Food-A*, we first train each network on the training split, consisting only of the original images. We divide the test split into three sets, 1,533, 1,533, and 1,534 images, respectively. The first one is used for $\mathcal{D}_s$ as is (i.e., without corruption). We apply the image corruption method to the second and third sets. To be specific, splitting the 19 corruption types into 6 and 13, we apply the 6 corruptions to the second set to make $\mathcal{D}_o$'s, and the 13 corruptions to the last to make $\mathcal{D}_t$'s. As each corruption has five severity levels, there are $30(= 6 \times 5)$ $\mathcal{D}_o$'s and $65(= 13 \times 5)$ $\mathcal{D}_t$'s. The former is used for training $f$ (precisely, 20 are used for training and 10 are for validation), and the latter is for evaluating $f$.

For ImageNet, we choose 5,000, 2,000, and 5,000 images from the validation split without overlap. We use them to make $\mathcal{D}_s$, $\mathcal{D}_o$'s, and $\mathcal{D}_t$'s, respectively. As with *Food-A*, we apply the 6 and 13 types of corruption to the second and third sets, making 30 $\mathcal{D}_o$'s and 65 $\mathcal{D}_t$'s, respectively.

For the evaluation of $f$, we calculate *mean absolute error (MAE)* and *root mean squared error (RMSE)* of the predicted $\overline{\mathrm{err}}$ over the 65 $\mathcal{D}_t$'s. We repeat this for 20 times with different splits of image corruptions $(19 \to 6 + 13)$, reporting their mean and standard deviation.

Table 3.3 shows the results for *Food-A* and ImageNet. (The accuracies of the original classification tasks of *Food-A* and ImageNet are reported in Table 3.7 and Table 3.10 in Sec. 3.5 and 3.6.) It is seen for both datasets that *Cosine* achieves the top-level accuracy irrespective of the training methods. For *Food-A*, using a pre-trained network boosts the performance for the confidence-based methods (i.e., from *Baseline* to *ODIN**), resulting in that *MC dropout* performs the best; *Cosine* attains almost the same accuracy. On the other hand, *Mahalanobis* and *PAD* do not perform well regardless of the datasets and training methods. This well demonstrates the difference between detecting the distributional shift $p(x) \to p'(x)$ and detecting the deterioration of classification accuracy. We show scatter plots of $\overline{S}$ vs. $\overline{\mathrm{err}}$ in Fig. 3.4 and 3.5 in Sec. 3.6, which provides a similar, or even clearer, observation.

**Office-31**

To study another type of domain shift, we employ the Office-31 dataset [88], which is popular in the study of domain adaptation. The dataset consists of three subsets, Amazon, DSLR, and Webcam, which share the same 31 classes and are collected from different domains. We train our CNNs on Amazon and evaluate the compared methods in terms of prediction accuracy of classification errors for samples in DSLR and Webcam. The classification accuracy of the CNNs on Amazon is provided in Table 3.11 in Sec. 3.7.

To obtain $\mathcal{D}_o$'s for training $f$, we employ the same image corruption methods as Sec. 3.3.3; we apply them to Amazon samples to create virtual domain-shifted samples. The effectiveness of modeling the true shifted data, i.e., DSLR and Webcam, with these samples is unknown and needs to be experimentally validated. If this works, it will be practically useful. Specifically, we split the test splits of Amazon containing 754 images evenly into two sets. We use one for $\mathcal{D}_s$ and the other for creating $\mathcal{D}_o$'s. We apply all the types of corruption, yielding $95(= 19 \times 5)$ $\mathcal{D}_o$'s. We then split them into those generated by four corruptions and those generated by the rest; the latter is used for training $f$, and the former is used for the validation. We iterate this for 20 times with different random splits of the corruption types, reporting the average over $20 \times 3$ trials, as there are three CNN models trained from different initial weights.

To evaluate each method (i.e., $f$ based on a OOD score), we split DSLR and Webcam into subsets containing 50 samples, yielding 18 $\mathcal{D}_t$'s in total. We apply $f$ to each of them, reporting the average error of predicting classification errors. Table 3.4 shows the results. It is observed that *Cosine* works well in both training methods. The two variants of *Mahalanobis* show good performance when using a pre-trained model, but this may be better considered a coincidence, as explained below. Figure 3.3 shows the scatter plots of OOD score vs. classification error for each method. The green dots indicate $D_o$'s, corrupted Amazon images used for training $f$, and the blue ones indicate $D_t$'s, subsets from DSLR and Webcam containing 50 samples each. For the method for which the green dots distribute with narrower spread, the regressor $f$ will yield more accurate results. Thus, it is seen from Fig. 3.3 that both *Mahalanobis*'s tend to have large spread, meaning that they could perform

Table 3.4: Errors of the predicted classification error by the compared methods on 50 sample subsets of DSLR and Webcam. The CNN is trained on Amazon and the regressor $f$ is trained using corrupted images of Amazon.

| Method | Train-from-scratch | | Fine-tuning | |
|---|---|---|---|---|
| | MAE | RMSE | MAE | RMSE |
| *Baseline* | 12.1(3.1) | 14.6(3.1) | 10.6(2.3) | 11.7(2.3) |
| *Calib.* | 9.5(3.2) | 11.5(3.4) | 9.7(2.3) | 10.8(2.3) |
| *MC dropout* | 8.0(1.9) | 10.1(2.7) | 9.3(1.7) | 10.5(1.7) |
| *Cosine* | **5.6(1.5)** | **6.8(1.7)** | 8.5(2.0) | 10.0(2.2) |
| *ODIN** | 7.3(2.5) | 8.1(2.6) | 13.4(3.6) | 15.3(3.6) |
| *Maha. (sum)* | 18.8(4.7) | 20.7(3.9) | **7.9(2.0)** | **9.7(2.1)** |
| *Maha. (adv)* | 34.1(15.8) | 40.0(22.5) | 8.2(2.1) | 9.9(2.2) |
| *PAD* | 10.6(2.2) | 12.1(2.6) | 16.8(3.4) | 18.3(3.2) |

poorly depending on incoming domain-shifted data. *Cosine* and *MC dropout* have narrower spread, confirming their performance in Table 3.4. Other results for DSLR and Webcam subsets with a different number of samples are provided in Sec. 3.7.

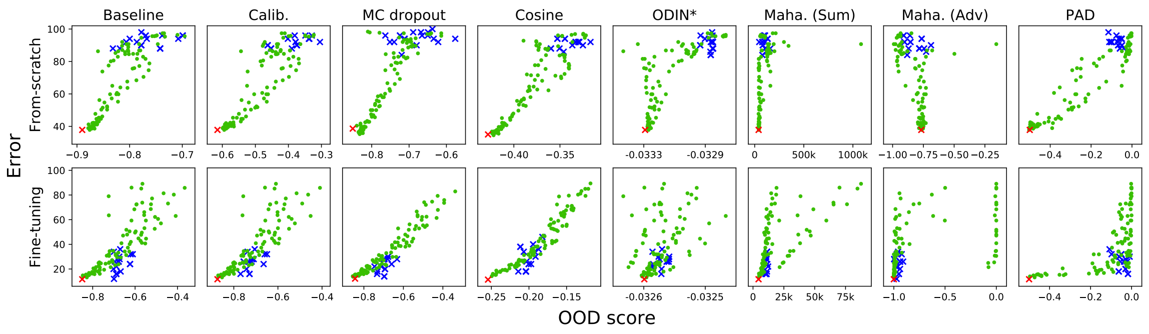


Figure 3.3: OOD score vs. the true classification error for 95($= 19\times5$) $D_o$'s (corrupted Amazon images used for training the regressor $f$; in green), 18 $D_t$'s (subsets of DSLR and Webcam containing 50 samples each; in blue), and $D_s$ (original Amazon images; in red).

### 3.3.4 Analyses of the Results

We can summarize our findings in the following. i) Using a pre-trained network has shown improvements in all the scenarios, confirming the report of [56]. ii) The detector using cosine similarity consistently works well throughout the three scenarios. The method will be the first choice if it is acceptable to modify the network's final layer. iii) The Mahalanobis detector, a SOTA method, works well only for irrelevant input detection. This is not contradictory with the previous reports, since they employ only this very scenario. The method fits a Gaussian distribution to ID samples belonging to each class and uses the same covariance matrix for all the classes. This strategy might work well on easy cases when incoming OOD samples are mapped distantly from the Gaussian distributions. However, such a simple modeling method will not work in more challenging cases. For instance, incoming OOD samples could be mapped near the ID distributions, as in novel class detection. In such cases, the ID sample distribution needs to be very precisely modeled, for which the assumption of Gaussian distributions with a single covariance matrix is inadequate. iv) Domain shift detection requires detecting classification accuracy deterioration, not detecting a distributional shift of inputs, contrary to its name. This theoretically favors the confidence-based methods; they (particularly MC dropout) indeed work well, when used with a pre-trained network. However, the Mahalanobis detector is more like an anomaly detection method, although its similarity with a softmax classifier is suggested in [4]. An input sample for which the network can make a correct classification can be detected as an 'anomaly' by the Mahalanobis detector.

## 3.4 Additional Results for Detection of Irrelevant Inputs

In our experiment for irrelevant input detection, using five datasets, we consider every pair of them, one for ID and the other for OOD. In Sec. 3.3.1, we reported only the average detection accuracy over four such pairs for an ID dataset. We report here the results for all the ID-OOD pairs. Tables 3.5 and 3.6 show the performance of the compared methods for training from scratch and for fine-tuning of a pre-trained network.

Table 3.5: The OOD detection performance (AUROC) for networks trained from scratch. D1=*Dog*, D2=*Plant*, D3=*Food*, D4=*Bird*, and D5=*Cat*.

| Method | OOD | In-Distribution | | | | |
|---|---|---|---|---|---|---|
| | | D1 | D2 | D3 | D4 | D5 |
| *Baseline* | D1 | - | 78.1(1.1) | 88.9(0.7) | 59.2(1.0) | 61.4(3.9) |
| | D2 | 76.1(12.4) | - | 63.6(4.5) | 32.7(5.6) | 70.8(21.1) |
| | D3 | 65.5(4.2) | 75.9(4.6) | - | 54.1(1.2) | 56.3(4.4) |
| | D4 | 72.1(3.2) | 74.8(2.1) | 88.0(0.8) | - | 61.2(4.4) |
| | D5 | 73.4(2.4) | 79.5(5.4) | 93.5(1.0) | 53.0(0.9) | - |
| *Calib.* | D1 | - | 76.1(1.3) | 91.6(0.6) | 65.1(0.6) | 67.2(3.5) |
| | D2 | 83.2(12.2) | - | 65.3(5.3) | 39.6(3.9) | 76.4(20.3) |
| | D3 | 70.4(5.4) | 73.6(7.6) | - | 58.2(1.5) | 60.3(5.6) |
| | D4 | 78.9(3.4) | 72.8(1.9) | 90.8(0.8) | - | 66.5(4.5) |
| | D5 | 81.4(3.0) | 77.7(6.4) | 95.7(0.8) | 54.4(1.7) | - |
| *MC dropout* | D1 | - | 82.6(2.8) | 89.3(0.2) | 61.2(1.1) | 67.0(1.9) |
| | D2 | 82.5(9.9) | - | 67.4(7.7) | 43.8(7.4) | 84.4(8.8) |
| | D3 | 68.6(1.6) | 84.7(2.6) | - | 57.1(1.6) | 61.3(2.8) |
| | D4 | 73.7(1.6) | 79.0(2.0) | 88.3(0.4) | - | 67.3(1.4) |
| | D5 | 75.9(2.4) | 84.6(4.6) | 94.7(0.0) | 57.0(0.7) | - |
| *Cosine* | D1 | - | 93.9(2.2) | 96.9(0.2) | 74.8(0.5) | 89.7(2.3) |
| | D2 | 96.6(1.8) | - | 94.1(0.5) | 51.1(13.9) | 98.1(1.4) |
| | D3 | 85.0(1.4) | 94.4(2.5) | - | 78.8(1.9) | 81.7(4.7) |
| | D4 | 90.9(0.7) | 94.1(1.4) | 97.6(0.1) | - | 90.6(2.7) |
| | D5 | 92.5(0.5) | 96.3(0.9) | 99.5(0.0) | 73.6(2.5) | - |
| *ODIN** | D1 | - | 55.8(16.6) | 74.2(0.9) | 54.5(2.1) | 63.5(2.1) |
| | D2 | 78.4(3.1) | - | 55.2(9.2) | 58.6(11.8) | 76.5(6.6) |
| | D3 | 69.3(6.1) | 54.9(21.9) | - | 62.5(3.0) | 68.7(2.0) |
| | D4 | 82.3(2.4) | 48.9(15.1) | 82.3(0.6) | - | 70.3(1.5) |
| | D5 | 81.2(3.8) | 57.0(18.1) | 89.6(0.7) | 51.6(2.7) | - |
| *Maha. (sum)* | D1 | - | 99.7(0.1) | 68.6(0.8) | 67.5(1.8) | 63.6(3.0) |
| | D2 | 87.8(3.4) | - | 97.8(0.6) | 75.7(4.5) | 92.7(5.9) |
| | D3 | 73.2(2.7) | 99.8(0.1) | - | 87.1(1.4) | 78.4(2.8) |
| | D4 | 75.9(1.2) | 99.7(0.1) | 87.2(0.8) | - | 76.2(2.0) |
| | D5 | 53.9(2.7) | 99.8(0.1) | 63.9(2.6) | 83.2(1.8) | - |
| *Maha. (adv)* | D1 | - | 80.9(10.8) | 65.6(3.1) | 59.8(3.6) | 35.2(8.2) |
| | D2 | 67.9(6.0) | - | 98.4(0.4) | 58.2(8.5) | 51.3(23.0) |
| | D3 | 41.0(4.2) | 87.0(8.2) | - | 59.2(3.5) | 39.5(9.7) |
| | D4 | 26.2(3.3) | 87.3(6.1) | 71.0(2.4) | - | 33.5(15.2) |
| | D5 | 35.5(6.6) | 70.9(19.4) | 41.1(6.6) | 74.2(9.7) | - |

Table 3.6: The OOD detection performance (AUROC) for fine-tuned networks from a pre-trained model. D1=*Dog*, D2=*Plant*, D3=*Food*, D4=*Bird*, and D5=*Cat*.

| Method | OOD | In-Distribution | | | | |
|---|---|---|---|---|---|---|
| | | D1 | D2 | D3 | D4 | D5 |
| *Baseline* | D1 | - | 96.3(0.4) | 91.4(0.5) | 96.4(0.6) | 99.3(0.5) |
| | D2 | 100.0(0.0) | - | 54.3(8.2) | 96.9(2.8) | 87.4(4.1) |
| | D3 | 99.7(0.0) | 96.1(0.7) | - | 99.2(0.2) | 97.1(0.6) |
| | D4 | 99.0(0.2) | 97.2(0.4) | 91.1(0.5) | - | 97.6(0.6) |
| | D5 | 100.0(0.0) | 95.2(1.5) | 95.9(1.2) | 99.8(0.1) | - |
| *Calib.* | D1 | - | 97.4(0.4) | 92.3(0.5) | 94.2(0.9) | 98.4(1.1) |
| | D2 | 100.0(0.0) | - | 55.0(8.3) | 92.9(6.0) | 73.0(6.2) |
| | D3 | 99.4(0.1) | 97.1(0.6) | - | 98.2(0.5) | 94.2(1.1) |
| | D4 | 98.1(0.5) | 98.1(0.3) | 91.8(0.5) | - | 94.7(1.1) |
| | D5 | 100.0(0.0) | 96.9(1.1) | 96.4(1.1) | 99.7(0.1) | - |
| *MC dropout* | D1 | - | 95.7(1.8) | 92.7(0.5) | 96.8(0.4) | 99.6(0.2) |
| | D2 | 100.0(0.0) | - | 61.4(7.3) | 99.3(0.4) | 88.0(7.1) |
| | D3 | 99.7(0.0) | 96.1(0.9) | - | 99.3(0.2) | 97.9(0.8) |
| | D4 | 98.9(0.1) | 97.2(0.7) | 91.5(0.7) | - | 98.7(0.5) |
| | D5 | 100.0(0.0) | 96.4(1.1) | 97.1(0.7) | 99.9(0.0) | - |
| *Cosine* | D1 | - | 99.3(0.5) | 96.3(0.0) | 97.5(0.2) | 99.4(0.0) |
| | D2 | 99.5(0.2) | - | 86.7(6.1) | 100.0(0.0) | 98.6(0.4) |
| | D3 | 99.5(0.1) | 99.5(0.2) | - | 99.4(0.1) | 99.0(0.3) |
| | D4 | 99.5(0.0) | 99.6(0.3) | 96.0(0.3) | - | 99.1(0.2) |
| | D5 | 99.8(0.0) | 99.8(0.1) | 99.2(0.2) | 99.6(0.0) | - |
| *ODIN** | D1 | - | 93.1(1.7) | 86.2(0.7) | 92.2(0.6) | 99.8(0.1) |
| | D2 | 99.6(0.2) | - | 53.5(16.5) | 98.0(0.9) | 91.8(2.8) |
| | D3 | 96.6(0.6) | 92.7(3.3) | - | 96.8(0.5) | 98.2(0.2) |
| | D4 | 97.2(0.6) | 95.7(1.3) | 90.0(0.3) | - | 99.6(0.1) |
| | D5 | 100.0(0.0) | 98.6(0.1) | 96.2(0.7) | 99.5(0.2) | - |
| *Maha. (sum)* | D1 | - | 100.0(0.0) | 93.2(0.5) | 99.0(0.1) | 99.6(0.0) |
| | D2 | 99.7(0.0) | - | 97.5(0.2) | 99.8(0.0) | 99.9(0.0) |
| | D3 | 98.9(0.0) | 100.0(0.0) | - | 99.3(0.1) | 99.8(0.0) |
| | D4 | 97.4(0.1) | 100.0(0.0) | 96.4(0.1) | - | 99.5(0.0) |
| | D5 | 98.1(0.0) | 100.0(0.0) | 92.8(0.4) | 99.1(0.0) | - |
| *Maha. (adv)* | D1 | - | 99.1(1.0) | 86.9(1.5) | 98.2(0.1) | 99.6(0.0) |
| | D2 | 99.7(0.0) | - | 82.7(6.2) | 99.7(0.0) | 100.0(0.0) |
| | D3 | 98.5(0.1) | 99.0(1.0) | - | 98.3(0.2) | 99.8(0.0) |
| | D4 | 92.2(1.1) | 98.9(1.2) | 92.7(1.1) | - | 99.5(0.0) |
| | D5 | 98.1(0.1) | 99.6(0.4) | 88.1(1.3) | 98.9(0.1) | - |

Table 3.7: Classification accuracy for the two tasks, *Dog* (20 dog breeds classification) and *Food-A* (46 food class classification), for which novel class detection is examined.

| Dataset | Train-from-scratch | | | Fine-tuning | | |
|---|---|---|---|---|---|---|
| | *Standard* | *MC dropout* | *Cosine* | *Standard* | *MC dropout* | *Cosine* |
| *Dog* | 51.9(0.8) | 51.1(0.8) | 64.2(1.4) | 95.7(0.1) | 95.6(0.2) | 96.0(0.1) |
| *Food-A* | 83.4(0.3) | 84.0(0.2) | 83.6(0.1) | 87.5(0.2) | 87.5(0.2) | 86.0(0.1) |

## 3.5 Additional Results for Detection of Novel Classes

### 3.5.1 Classification Accuracy of the Base Tasks

In our experiments for novel class detection, we employ two datasets, *Dog* and *Food-A*. Table 3.7 shows the classification accuracy for each of them. It is seen that for *Dog*, using a pre-trained model boosts the accuracy. There is a tendency similar to that seen in Table 3.1, that *Cosine* outperforms others in training from scratch. For *Food-A*, using a pre-trained model shows only modest improvement due to the availability of a sufficient number of samples.

### 3.5.2 Additional Results

In one of the experiments explained in Sec. 3.3.2, we use only dog classes from the Oxford-IIIT Pet dataset. We show here additional results obtained when using cat classes. Choosing nine from 12 cat breeds contained in the dataset, we train the networks on classification of these nine breeds and test novel class detection using the remaining three breed classes. In another experiment, we use *Food-A* for ID and *Food-B* for OOD. We report here the results for the reverse configuration. Table 3.8 shows the classification accuracy of the new tasks. Table 3.9 shows the performance of the compared methods on the novel class detection. A similar observation to the experiments of Sec. 3.3.2 can be made.

Table 3.8: Classification accuracy for *Cat* (9 cat breed classification) and *Food-B* (47 food class classification).

| Dataset | Random Init. | | | Fine-Tuning | | |
|---|---|---|---|---|---|---|
| | *Standard* | *MC dropout* | *Cosine* | *Standard* | *MC dropout* | *Cosine* |
| *Cat* | 60.9(0.8) | 57.8(1.2) | 64.1(0.9) | 89.5(0.3) | 88.8(0.3) | 88.9(0.5) |
| *Food-B* | 83.6(0.3) | 84.3(0.4) | 83.7(0.3) | 87.6(0.2) | 87.4(0.1) | 86.2(0.1) |

Table 3.9: Novel class detection performance (AUROC) of the compared methods. The OOD samples for *Cat* and *Food-B* are the held-out 3 cat breeds and *Food-A*, respectively.

| Method | *Cat* | | *Food-B* | |
|---|---|---|---|---|
| | From-scratch. | Fine-Tune | From-scratch | Fine-Tune |
| *Baseline* | 57.7(0.1) | 72.1(1.9) | 81.5(0.2) | 84.2(0.2) |
| *Calib.* | 58.6(0.4) | 70.9(2.0) | **82.5(0.2)** | 84.5(0.2) |
| *MC dropout* | 57.6(1.3) | 72.7(1.1) | 82.0(0.2) | 84.5(0.3) |
| *Cosine* | **63.6(1.1)** | **73.7(2.1)** | 81.7(0.2) | **84.8(0.3)** |
| *ODIN** | 52.3(0.7) | 72.6(0.7) | 74.4(0.7) | 73.6(0.3) |
| *Maha. (sum)* | 43.9(0.5) | 62.2(0.4) | 51.8(0.2) | 64.2(0.1) |
| *Maha. (adv)* | 50.0(3.0) | 61.1(2.1) | 52.6(0.4) | 52.8(9.8) |

Table 3.10: Classification accuracy on ImageNet for each network.

| Dataset | *Standard* | *MC dropout* | *Cosine* |
|---|---|---|---|
| ImageNet | 74.6(0.1) | 75.5(0.3) | 72.4(0.1) |

## 3.6 Additional Results for Detection of Domain Shift (Image Corruption)

### 3.6.1 Classification Accuracy on ImageNet

Table 3.10 shows the accuracy of the three networks used by the compared OOD detection methods for 1,000 class classification of the ImageNet dataset. We use center-cropping at test time. The cosine network shows lower classification accuracy here.

### 3.6.2 Scatter Plots of OOD Score vs. Classification Error

In Sec. 3.3.3, we showed experimental results of domain shift detection using *Food-A*. Given a set $\mathcal{D}_t$ of samples, each of the compared methods calculates an OOD score $\overline{S}$ for it, from which the average classification error $\overline{\mathrm{err}}$ over samples from $\mathcal{D}_t$ is predicted. Figure 3.4 shows scatter plots showing the relation between the OOD score $\overline{S}$ and the true classification error for a number of datasets (i.e., $\mathcal{D}_t$'s). We have $95(= 19\times5)$ such datasets, each containing images undergoing one of the combinations of 19 image corruptions and 5 severity levels. The method with a narrower spread of dots should provide a more accurate estimation. These scatter plots well depict which method works well and which does not, which agrees well with Table 3.3. The same holds true for the plots for ImageNet shown in Fig. 3.5.

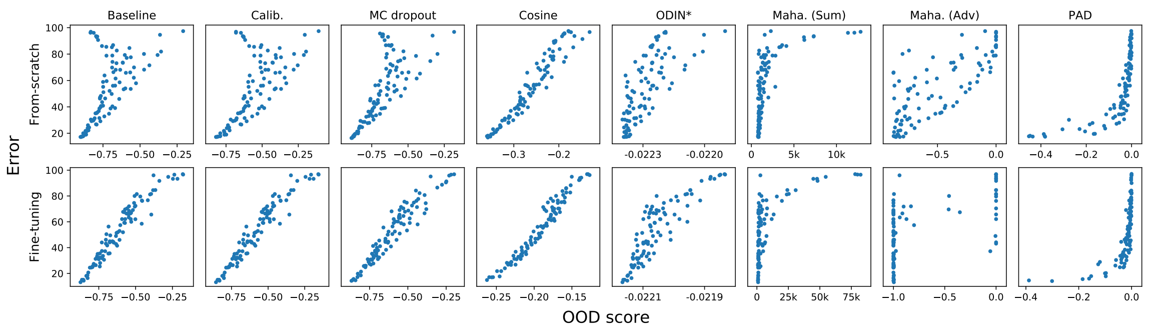


Figure 3.4: OOD score vs. classification error for 95(= $19 \times 5$) datasets, i.e., $D_o$'s and $D_t$'s (corrupted Food-A images).

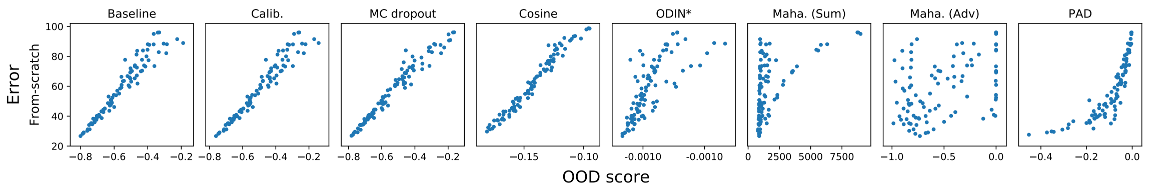


Figure 3.5: OOD score vs. classification error for 95(= $19 \times 5$) datasets, i.e., $D_o$'s and $D_t$'s (corrupted ImageNet images).

## 3.7 Additional Results for Detection of Domain Shift (Office-31)

### 3.7.1 Classification Accuracy of the Base Tasks

Table 3.11 shows the classification accuracy of the three networks used by the compared methods for the different domain datasets of Office-31. These networks are trained only on Amazon.

### 3.7.2 Additional Results

As with the experiments on image corruption, we evaluate how accurately the compared methods can predict the classification error on incoming datasets, $\mathcal{D}_t$'s. Table 3.4 and Fig. 3.3 show the error of the predicted classification accuracy and the scatter plots of the OOD score and the true classification accuracy, where $\mathcal{D}_t$'s are created by splitting DSLR and Webcam into sets containing 50 samples. We show here additional results obtained for $\mathcal{D}_t$'s created differently. Table 3.12 and Fig. 3.6

Table 3.11: The classification accuracy of the three networks trained on Amazon for Amazon, DSLR, and Webcam.

| Dataset | Train-from-scratch | | | Fine-tuning | | |
|---|---|---|---|---|---|---|
| | *Standard* | *MC dropout* | *Cosine* | *Standard* | *MC dropout* | *Cosine* |
| Amazon | 63.0(1.3) | 63.9(2.5) | 67.1(1.8) | 87.8(0.7) | 87.1(0.1) | 87.8(0.3) |
| DSLR | 9.6(0.6) | 7.8(1.6) | 10.0(1.0) | 77.1(1.2) | 78.9(1.3) | 74.2(0.9) |
| Webcam | 7.2(1.9) | 6.8(1.2) | 11.1(0.6) | 73.8(1.3) | 74.1(2.2) | 67.8(0.3) |

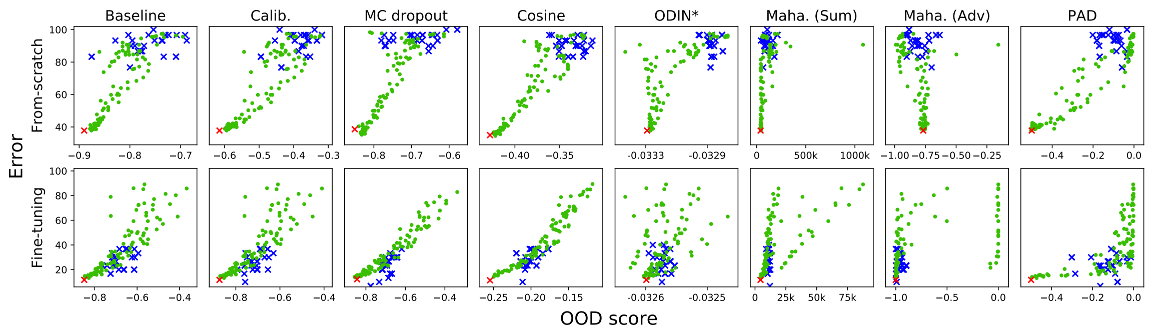


Figure 3.6: OOD score vs. classification error for 95(= 19 × 5) $D_o$'s (corrupted Amazon images used for training the regressor $f$; in green), 32 $D_t$'s (subsets of DSLR and Webcam containing 30 samples each; in blue), and $D_s$ (original Amazon images; in red).

show the prediction errors and the scatter plots for $\mathcal{D}_t$'s containing 30 samples. Table 3.13 and Fig. 3.7 show those for $\mathcal{D}_t$'s of 100 samples. Table 3.14 and Fig. 3.8 show those for using the entire DSLR and Webcam for $\mathcal{D}_t$'s; thus there are only two $\mathcal{D}_t$'s. The standard deviations are computed for 20 × 3 trials (20 for random splitting of corruption types for train/val and 3 for network models trained from random initial weights), as explained in Sec. 3.3.3.

## 3.8 Effectiveness of Ensembles

An ensemble of multiple models is known to performs better than MC-dropout we considered in the main experiments for estimation of uncertainty etc. It is also known

Table 3.12: Errors of the predicted classification error by the compared methods on 30 sample subsets of DSLR and Webcam. The CNN is trained on Amazon and the regressor $f$ is trained using corrupted images of Amazon.

| Method | Train-from-scratch | | Fine-tuning | |
|---|---|---|---|---|
| | MAE | RMSE | MAE | RMSE |
| *Baseline* | 13.3(2.2) | 17.5(2.7) | 10.9(2.0) | 12.8(2.2) |
| *Calib.* | 9.8(2.4) | 12.6(2.7) | 9.9(2.1) | 11.9(2.3) |
| *MC dropout* | 9.7(2.4) | 12.1(3.5) | 10.0(1.6) | 11.7(1.6) |
| *Cosine* | **6.7(1.2)** | **8.3(1.5)** | 9.5(1.8) | 11.5(1.8) |
| *ODIN** | 8.1(2.4) | 9.3(2.6) | 13.5(3.6) | 16.1(3.8) |
| *Maha. (sum)* | 19.5(3.8) | 22.0(2.9) | **9.1(1.9)** | **11.2(2.0)** |
| *Maha. (adv)* | 33.6(15.7) | 40.0(22.5) | 9.2(1.8) | 11.3(1.9) |
| *PAD* | 13.1(2.9) | 14.7(3.2) | 13.9(2.3) | 16.2(2.3) |

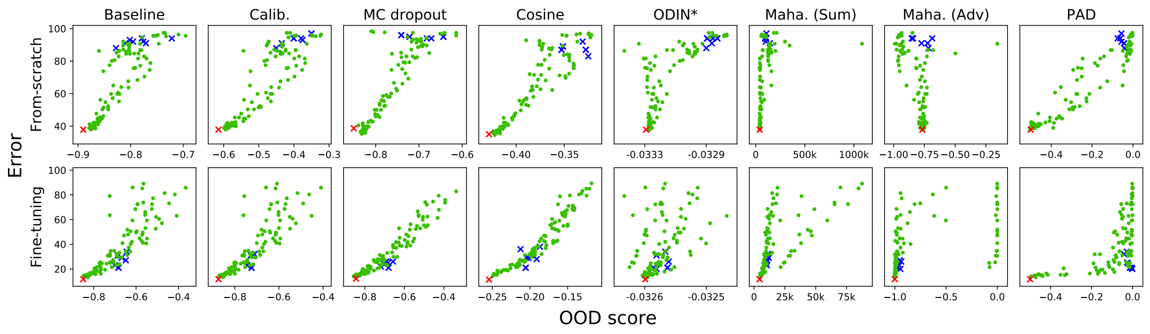


Figure 3.7: OOD score vs. classification error for 95(= 19 × 5) $D_o$'s (corrupted Amazon images used for training the regressor $f$; in green), 8 $D_t$'s (subsets of DSLR and Webcam containing 100 samples each; in blue), and $D_s$ (original Amazon images; in red).

Table 3.13: Errors of the predicted classification error by the compared methods on 100 sample subsets of DSLR and Webcam. The CNN is trained on Amazon and the regressor $f$ is trained using corrupted images of Amazon.

| Method | Train-from-scratch | | Fine-tuning | |
|---|---|---|---|---|
| | MAE | RMSE | MAE | RMSE |
| *Baseline* | 11.1(3.2) | 12.9(3.4) | 10.2(2.7) | 11.0(2.6) |
| *Calib.* | 8.7(3.2) | 10.4(3.7) | 9.2(2.7) | 10.0(2.6) |
| *MC dropout* | 6.9(2.5) | 8.2(2.9) | 8.8(1.9) | 9.7(2.0) |
| *Cosine* | **4.6(1.6)** | **5.5(1.7)** | 7.9(2.1) | 9.0(2.3) |
| *ODIN** | 6.8(3.1) | 7.3(3.1) | 13.4(3.8) | 15.1(3.7) |
| *Maha. (sum)* | 18.3(4.7) | 19.9(3.9) | **7.4(2.4)** | **8.5(2.4)** |
| *Maha. (adv)* | 33.6(15.3) | 39.1(22.0) | 7.7(2.2) | 8.7(2.2) |
| *PAD* | 8.6(2.3) | 9.3(2.2) | 18.7(3.7) | 19.3(3.7) |

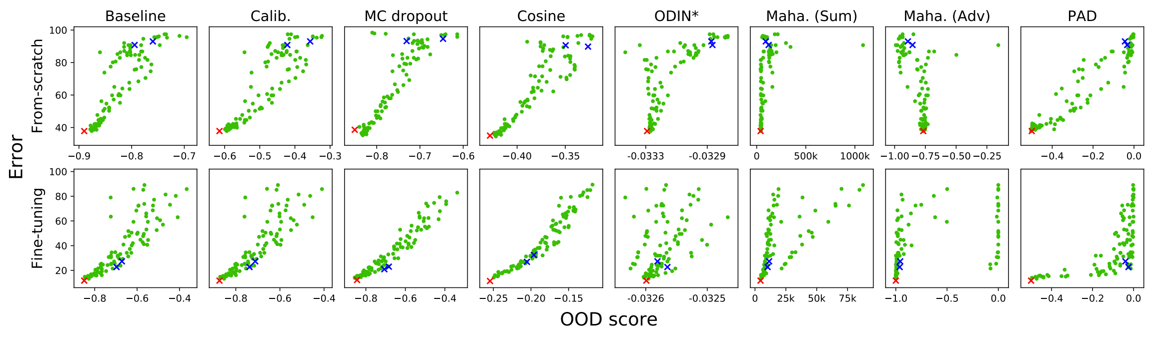


Figure 3.8: OOD score vs. classification error for 95(= 19 × 5) $D_o$'s (corrupted Amazon images used for training the regressor $f$; in green), 2 $D_t$'s (the entire set of DSLR and Webcam; in blue), and $D_s$ (original Amazon images; in red).

Table 3.14: Errors of the predicted classification error by the compared methods on the entire set of DSLR and Webcam. The CNN is trained on Amazon and the regressor $f$ is trained using corrupted images of Amazon.

| Method | Train-from-scratch | | Fine-tuning | |
|---|---|---|---|---|
| | MAE | RMSE | MAE | RMSE |
| *Baseline* | 9.1(2.1) | 10.4(2.5) | 10.0(2.1) | 10.1(2.1) |
| *Calib.* | 8.3(2.9) | 9.6(3.5) | 9.0(2.3) | 9.0(2.2) |
| *MC dropout* | 6.9(2.9) | 7.8(3.5) | 9.2(1.4) | 9.2(1.4) |
| *Cosine* | **4.0(1.8)** | **4.6(1.8)** | **7.5(2.2)** | **7.5(2.2)** |
| *ODIN** | 6.5(3.0) | 6.7(3.0) | 12.7(3.5) | 14.1(3.7) |
| *Maha. (sum)* | 17.8(4.8) | 19.2(3.7) | **7.5(1.9)** | 7.6(1.9) |
| *Maha. (adv)* | 27.7(14.0) | 32.9(20.9) | 7.7(1.9) | 7.9(1.9) |
| *PAD* | 6.3(1.9) | 6.5(1.8) | 19.1(2.5) | 19.3(2.6) |

to be better approximation to Bayesian networks. Thus, we experimentally evaluate ensembles. We consider an ensemble of five models and train each model in two ways, i.e., "from-scratch" and "fine-tuning." We randomly initialize all the weights of each model for the former. We initialize the last layer randomly and other layers with the pre-trained model's weights for the latter. We evaluate ensembles for Baseline and Cosine. Tables 3.15, 3.16, 3.17, and 3.18 show the results for the three scenarios. In the tables, "(con.)" means confidence is used as an ID score, or equivalently, negative confidence is used as an OOD score. "(en.)" means the entropy is used as an OOD score.

We can observe the following from the tables:

- An ensemble of models performs better than a single model. This is always true for Baseline. The same is true for Cosine except for domain shift detection. (The reason is not clear.)

- An ensemble of Baseline models still performs lower than a single Cosine model

Table 3.15: Irrelevant input detection performance of the ensemble models.

| Method | From-scratch | Fine-Tune |
|---|---|---|
| *Baseline (con.)* | 61.4(12.1) | 97.7(3.1) |
| *Ensemble (con.)* | 67.8(13.7) | 98.3(2.2) |
| *Baseline (en.)* | 64.8(13.7) | 99.2(0.9) |
| *Ensemble (en.)* | 73.4(15.3) | 99.5(0.5) |
| *Cosine* | 83.9(11.4) | 99.0(0.7) |
| *Ensemble cosine* | 85.7(12.9) | 99.1(0.7) |

Table 3.16: Novel class detection performance of the ensemble models.

| Method | Dog | | Food-A | |
|---|---|---|---|---|
| | From-scratch | Fine-Tune | From-scratch | Fine-Tune |
| *Baseline (con.)* | 61.1(0.8) | 88.7(1.0) | 82.5(0.1) | 84.6(0.2) |
| *Ensemble (con.)* | 64.7 | 89.5 | 84.3 | 86.0 |
| *Baseline (en.)* | 61.6(0.7) | 90.0(1.0) | 83.3(0.1) | 85.4(0.2) |
| *Ensemble (en.)* | 65.7 | 90.8 | 85.0 | 86.8 |
| *Cosine* | 68.8(1.3) | 94.1(0.8) | 83.7(0.1) | 85.7(0.3) |
| *Ensemble cosine* | 72.0 | 94.4 | 85.2 | 86.8 |

for most cases. It sometimes shows better performance for fine-tuned models, but the margin is small.

- Using entropy as OOD score tends to show slightly better performance than using confidence.

We conclude that Cosine's superiority remains true even when we take ensembles into consideration.

Table 3.17: Errors of the predicted classification error by the ensemble models.

| Method | *Food-A (From-scratch)* | | *Food-A (Fine-tuning)* | | ImageNet | |
|---|---|---|---|---|---|---|
| | MAE | RMSE | MAE | RMSE | MAE | RMSE |
| *Baseline (con.)* | 15.8(3.0) | 20.5(3.7) | 6.4(1.3) | 7.9(1.6) | 4.6(0.8) | 6.3(1.0) |
| *Ensemble (con.)* | 12.9(2.3) | 17.1(2.4) | 5.6(1.3) | 7.0(1.8) | 4.0(0.8) | 5.5(1.1) |
| *Baseline (en.)* | 16.8(3.2) | 21.6(3.6) | 6.6(1.0) | 8.4(1.3) | 4.7(0.8) | 6.7(1.1) |
| *Ensemble (en.)* | 14.6(2.0) | 19.3(2.5) | 6.0(0.9) | 7.5(1.3) | 3.9(0.4) | 5.7(0.6) |
| *Cosine* | 6.6(1.3) | 8.2(1.6) | 6.1(1.6) | 7.5(2.2) | 3.8(0.9) | 4.7(1.1) |
| *Ensemble cosine* | 7.3(1.3) | 9.0(1.4) | 6.4(1.6) | 8.0(2.2) | 4.2(1.0) | 5.2(1.3) |

Table 3.18: Errors of the predicted classification error by the ensemble models on 50 sample subsets of DSLR and Webcam.

| Method | Train-from-scratch | | Fine-tuning | |
|---|---|---|---|---|
| | MAE | RMSE | MAE | RMSE |
| *Baseline (con.)* | 12.1(3.1) | 14.6(3.1) | 10.6(2.3) | 11.7(2.3) |
| *Ensemble (con.)* | 10.4(2.2) | 11.8(2.2) | 9.0(1.1) | 10.9(1.2) |
| *Baseline (en.)* | 11.5(2.5) | 12.7(2.4) | 11.4(2.9) | 13.7(2.9) |
| *Ensemble (en.)* | 11.2(2.2) | 12.6(2.1) | 7.5(1.0) | 8.7(1.1) |
| *Cosine* | 5.6(1.5) | 6.8(1.7) | 8.5(2.0) | 10.0(2.2) |
| *Ensemble cosine* | 5.3(1.1 | 7.1(1.5) | 8.6(1.6) | 10.3(1.8) |

Table 3.19: Specifications of the datasets used in the experiments.

| Dataset | # of classes | # of samples | Ave. image size | Used in |
|---|---|---|---|---|
| Dog Breeds | 120 | 10,222 | 443 × 387 | Section 3.1 |
| Plant Seeding | 12 | 5,544 | 357 × 356 | Section 3.1 |
| Food-101 | 101 | 101,000 | 496 × 475 | Sections 3.1, 3.2, and 3.3.2 |
| CUB-200 | 200 | 11,788 | 468 × 386 | Section 3.1 |
| Stanford Car | 196 | 16,185 | 700 × 483 | Section 3.1 |
| Oxford-IIIT Pet | 37 | 7,393 | 437 × 391 | Section 3.2 |
| ImageNet | 1000 | 1,281,167 | 482 × 418 | Section 3.3.2 |
| Office-31 | 31 | 4,110 | 418 × 418 | Section 3.3.3 |

## 3.9 Additional Details of Experimental Settings

### 3.9.1 Training of the Networks

As is mentioned in Sec. 3.3, we employ Resnet-50 in all the experiments. For the optimization, we use SGD with the momentum set to 0.9 and the weight decay set to $10^{-4}$. The learning rate starts at 0.1, and then is divided by 10 depending on the performance of the validation dataset.

To fine-tune a pre-trained network, we use the learning rate of 0.001 for the standard network and that with MC dropout. For the network used with *Cosine*, we use the learning rate of 0.001 to the backbone part and a higher learning rate of 0.1 to the fully-connected layer; the weight decay for the fully-connected layer is set to 0, following [74] and [5].

### 3.9.2 Datasets

Table 3.19 shows the specification of the datasets used in our experiments. Note that we modify some of the dataset and use them in several experiments. In the experiments of domain shift detection, we employed image corruption to simulate/model domain shift. The example of the corrupted images are shown in Fig. 3.9.

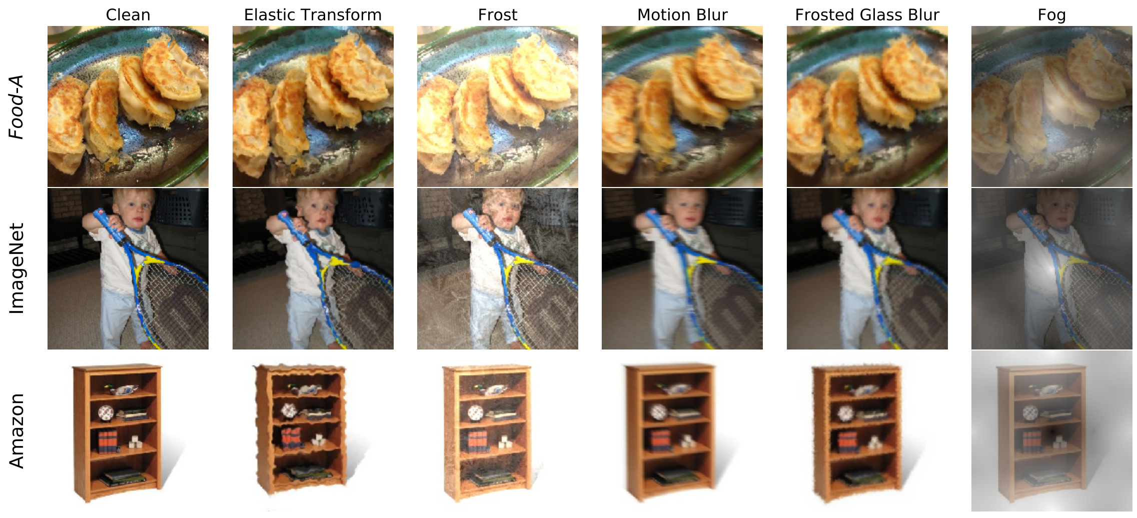


Figure 3.9: Examples of the corrupted images obtained by applying different types of image corruption [6].

## 3.10 Related Work

Many studies of OOD detection have been conducted so far, most of which are proposals of new methods; those not mentioned above include [25–27, 38, 89]. Experimental evaluation similar to our study but on the estimation of the uncertainty of prediction is provided in [18].

In [5], the authors present a scheme for conceptually classifying domain shifts in two axes, semantic shift and non-semantic shift. Semantic shift (S) represents OOD samples coming from the distribution of an unseen class, and non-semantic shift (NS) represents to OOD samples coming from an unseen domain. Through the experiments using the DomainNet dataset [90], they conclude that OOD detection is more difficult in the order of S $>$ NS $>$ S+NS.

In this study, we classify the problems into three types from an application perspective. One might view this as somewhat arbitrary and vague. Unfortunately, Hsu et al.'s scheme does not provide help. For instance, according to their scheme, novel class detection is S, and domain shift is NS. However, it is unclear which to classify irrelevant detection between S and S+NS. Moreover, their conclusion (i.e., S $>$ NS $>$ S+NS) does not hold for our results; the difficulty depends on the closeness between classes and between domains. After all, we think that only applications can determine what constitutes domain and what constitutes classes. Further discussion will be left for a future study.

As mentioned earlier, the detection of domain shift in the context of deep learning has not been well studied in the community. The authors are not aware of a study for image classification and find only a few [75] even when looking at other fields. On the other hand, there are a large number of studies of *domain adaptation* (DA); [77, 78, 91–93] to name a few. It is to make a model that has learned a task using the dataset of a particular domain adapt to work on data from a different domain. Researchers have been studied several problem settings, e.g., closed-set, partial, open-set, and boundless DA [92]. However, these studies all assume that the source and target domains are already known; no study considers the case where the domain of incoming inputs is unidentified. Thus, they do not provide a hint of how to detect domain shift.

## 3.11 Summary and Conclusion

In this chapter, we first classified OOD detection into three scenarios from an application perspective, i.e., irrelevant input detection, novel class detection, and domain shift detection. We have presented a meta-approach to be used with any OOD detection method to domain shift detection, which has been poorly studied in the community. We have experimentally evaluated various OOD detection methods on these scenarios. The results show the effectiveness of the above approach to domain shift several as well as several findings such as which method works on which scenario.

# Chapter 4

# Bridging In- and Out-of-distribution Samples for Their Better Discriminability

## 4.1 Introduction

Detecting out-of-distribution (OOD) samples, i.e., samples from a distribution other than the distribution of the samples used for training (called 'in-distribution (ID)' samples), is a vital problem to cope with when deploying neural networks in real-world applications. There are many studies on the problem so far. The difficulty with the OOD detection lies in the requirement to distinguish ID and OOD samples by learning only ID samples. A natural approach is to formulate the problem as anomaly detection. Several methods [4, 25, 26, 89] utilize the intermediate layer activation of a network that is trained on ID samples; they model the distribution of ID samples in the feature space and detect OOD samples as anomalies. They perform fairly well in some cases but show limitations [1, 5, 74].

To gain further performance, it seems necessary to have a better feature space, in which OOD samples are more clearly distinguished from ID samples. Evidence for the necessity is that using a network pre-trained on a large dataset (e.g., ImageNet) makes OOD detection easier [56]. Furthermore, a method using cosine similarity to model class probabilities, as with metric learning methods, has been proposed [5, 74], showing promising results. There are several methods [3, 25–27] that map the layer activation from a trained network into a good feature to enable more accurate OOD detection.

Outlier Exposure (OE) [7] is yet another approach to OOD detection that can be thought of as aiming at the same goal. It uses an available OOD dataset at training time, which hypothetically does not need to match the OOD sample distribution we will encounter in practice. A network is trained to classify an input to its true class if it is ID and 'none of the classes,' (i.e., identical probabilities for all the ID classes) if it is OOD. It aims to learn a better internal representation that helps identify unseen OOD samples accurately. While it shows good experimental performance, its success inevitably depends on the (dis)similarity between the assumed and the real OOD samples, which is hard to quantify in experiments; thus, its real-world performance remains unclear.

In this chapter, we question the premise of previous studies that ID and OOD samples are separated distinctly, proposing a new OOD detection method. As mentioned above, OE regards any sample as either an ID or an OOD sample in an exclusive manner. In contrast, we consider samples in the intermediate between the two, i.e., those having an OOD likelihood between 0 to 100%. We consider these samples to have soft labels and train a network using them.

The problem is how to get such intermediate samples as well as their soft labels. To do this, we apply synthetic image corruption to ID samples, creating new samples lying between ID and OOD. The underlying thought is that applying very severe image corruption to ID samples will make them turn to OOD, as their semantic contents will be lost. On the other hand, less severe corruption will create samples maintaining their contents; their ID/OOD likelihoods will be in the range of 0 to 1.

To provide soft labels for these intermediate samples, we use a network trained on the ID samples alone in the standard fashion. Specifically, we apply an image corrupting transformation to all the samples of the ID training set and then input the transformed samples to the above network, calculating their mean classification accuracy. We then use it to create a soft label. Concretely, we design the soft label for a sample to have the mean classification accuracy as the probability of its true class and a constant probability for all other classes. This method creates a single

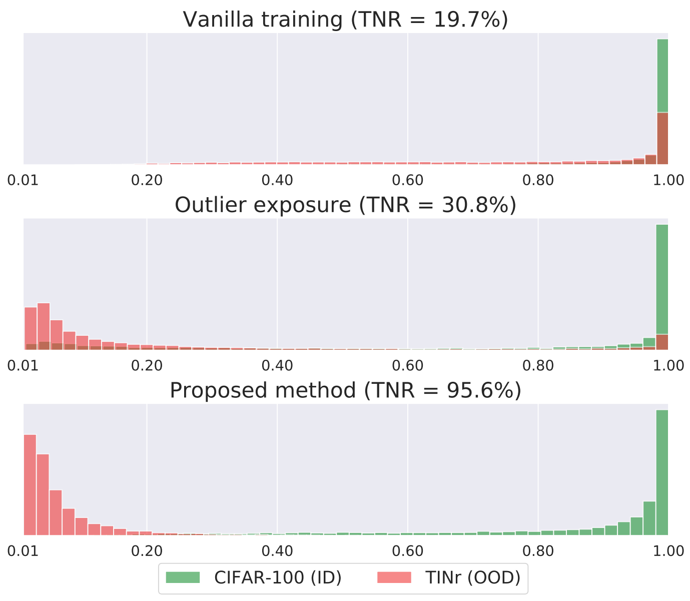


Figure 4.1: Histogram of the predicted confidence (i.e., maximum softmax probability) by a network (i.e., Wide Resnet 40-4) for in-distribution (ID) and out-of-distribution (OOD) samples. First row: The network with the vanilla training. Second row: Outlier exposure [7]. Third row: Proposed method. ID and OOD are CIFAR-100 and the resized Tiny ImageNet (TINr). *TNR at TPR 95%* are shown in the parentheses.

soft label for a single corrupting transformation. Ideally, we want to create soft labels distributing uniformly in the range between ID and OOD. For this purpose, we employ image corrupting methods that were developed to create the ImageNet-C dataset [6], which consists of fifteen different types of image corruption, each with five severity levels, i.e., 75 corruption methods in total. Leveraging their diversity, we create multiple soft labels sampling as densely as possible in the intermediate region between ID and OOD.

Training a standard CNN using the generated training samples with soft labels along with the original ID samples makes the CNN learn an improved internal representation, which separates ID samples and unseen OOD samples more clearly. Figure 4.1 shows the distributions of ID samples and OOD samples in the space of the predicted confidence of the same network trained differently. As a result, our method achieves state-of-the-art performance in the standard benchmark tests of OOD detection.

## 4.2 Related Work

### 4.2.1 OOD Detection

The maximum softmax probability (MSP), also called confidence, can be thought of as an estimate of the prediction's uncertainty. Using it with simple thresholding is a strong baseline for OOD detection [2], and many studies have proposed its extensions to improve detection accuracy. [3, 7, 20, 27, 38].

It is shown that the addition of a small perturbation to input that maximizes the confidence improves OOD detection accuracy [3, 5]. A method adding a separated branch to the network learning to predict the confidence of prediction is proposed [20]. It is also proposed to use the cosine similarity to compute logits instead of the ordinary linear transformation before softmax function [5, 74], yielding high performance. Other studies consider an ensemble of networks [17, 38], or considers a different problem setting [27].

Another group of methods formulates OOD detection as anomaly detection. They model the distribution of normal data (i.e., ID samples) in the space of some feature, which is intermediate layer activation [4] or its transformation by some mapping [25, 26, 89]. They then detect outliers of the distribution as OOD.

Some of the above methods (e.g., [3, 4, 89]) assume the accessibility to the true OOD samples if only a few. These methods have a few hyperparameters, which often significantly impact the final performance; thus, they determine them using the available OOD samples. Recent studies question this approach, as the true OOD samples are usually not accessible in practice [1, 5, 25–27, 74, 94].

There are also studies treating OOD detection more like anomaly detection; they do not use the model that has learned the ID class classification task. These studies model the distribution of ID samples using generative models, such as GAN [21], PixelCNN [55], and Normalizing Flow [22]. In [24], the authors synthetically blur input images and use the Random Network Distillation [95] to detect the OOD. However, these methods generally show inferior performance than the above methods that utilize the representation learned through the ID task training.

Yet another approach to detect OOD samples is to use the uncertainty of the prediction. We can think of the confidence-based methods mentioned above as following this approach. Methods with a more solid theoretical foundation are those based on Bayesian neural networks [13, 47, 48, 96]. However, these approaches do not show competitive performance to the above methods.

### 4.2.2 Soft Labels

In the standard setting of multi-class classification, target labels are represented as hard labels and used for training. While this is reasonable considering the nature of classification tasks, researchers have employed soft target labels for several purposes. One is the label smoothing. Since it was introduced to train Inception-v2 [97, 98], many studies have employed this trick, aiming at performance improvement; [99, 100], to name a few. Recently, Müller et al. [101] showed detailed analyses of the (in)effectiveness of label smoothing. Another use of soft target labels is seen in Knowledge Distillation [37]. A student network learns the soft labels provided by its teacher. It is shown in [101] that training the teacher with label smoothing worsens the student's performance. Soft labels are also used in the methods for dealing with label noise. Several methods estimate the confusion matrix defined between the network prediction and the provided noisy labels [102]. Others estimate the true labels of training samples during the training of a network [103, 104]. We may think these methods train networks using soft target labels. Some methods for data augmentation also use soft target labels. An example is Mixup [105], which interpolate two training samples by computing the weighted sum of not just inputs but their labels, yielding soft labels.

## 4.3 Proposed Method

### 4.3.1 Revisiting Outlier Exposure

Outlier Exposure (OE) is a method for OOD detection proposed by Hendrycks et al. [7]. It uses available OOD datasets for the training of a model to increase its sensitivity to unseen OOD samples. It is a general framework and they consider multi-class classification and density estimation. We consider the former here.

The method considers three distributions of samples, $\mathcal{D}_{\text{in}}$, $\mathcal{D}_{\text{out}}$, and $\mathcal{D}_{\text{out}}^{\text{OE}}$. Samples from $\mathcal{D}_{\text{in}}$ are in-distribution (ID). Samples from other distributions are OOD. $\mathcal{D}_{\text{out}}$ is an *unknown* distribution for OOD samples, which we will encounter at test time. $\mathcal{D}_{\text{out}}^{\text{OE}}$ is a *known* distribution of OOD samples, but it is unknown how similar to or different from $\mathcal{D}_{\text{out}}$ it is. A particular dataset is assumed to be available for $\mathcal{D}_{\text{out}}^{\text{OE}}$ and utilized for training the model. The idea of the method is to train a model using samples from both $\mathcal{D}_{\text{in}}$ and $\mathcal{D}_{\text{out}}^{\text{OE}}$ so that the model classifies the $\mathcal{D}_{\text{in}}$ samples correctly, whereas it predicts a uniform probability distribution for all the classes for $\mathcal{D}_{\text{out}}^{\text{OE}}$ samples. Specifically, denoting the model by $\hat{y} = f(x)$ for $K$-class classification, the method minimizes the following loss:

$$\mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{in}}}[\text{CE}(f(x), y)] + \lambda\mathbb{E}_{x\sim\mathcal{D}_{\text{out}}^{\text{OE}}}[\text{CE}(f(x), \mathcal{U}_K)], \tag{4.1}$$

where CE is cross-entropy; $y$ is the target distribution represented as a hard label (i.e., a one-hot vector) of the true class; $\mathcal{U}_K$ represents uniform distribution over $K$ classes, i.e., $y = [1/K, 1/K, \cdots, 1/K]$. The method aims to "learn a more conservative concept of the ID samples and enable the detection of novel forms of anomalies".

Their paper reports experimental results that show good performance of the method. The major issue with the method is that it is unclear how to specify $\mathcal{D}_{\text{out}}^{\text{OE}}$ (or precisely a dataset for $\mathcal{D}_{\text{out}}^{\text{OE}}$). It easy to specify an arbitrary dataset, but it is not guaranteed to lead to good results. Although it is formally distinguished from $\mathcal{D}_{\text{out}}$ in their experiments, the result will inevitably depend on their similarity. It is hard to examine how their (dis)similarity affects the performance. Therefore, it remains unclear if the method works well for various problems in the real world.

### 4.3.2 Corrupted Images as OOD Samples

It is ideal not to assume any specific OOD distribution or dataset. The problem is how to achieve good performance without it. An approach is to synthesize OOD samples from ID samples. Indeed, several previous studies [4, 89] propose to create adversarial examples from ID samples and use them as imaginary OOD samples. However, they only use the created samples to adjust hyperparameters to maximize OOD detection performance; the adversarial examples are not used for the training of networks.

Instead, we consider using corrupting and distorting images to synthesize OOD samples. Specifically, we corrupt ID samples in various ways, as shown in Fig. 4.2, and regard the corrupted images as OOD samples. The underlying thought is that severely corrupted ID images will become OOD samples.

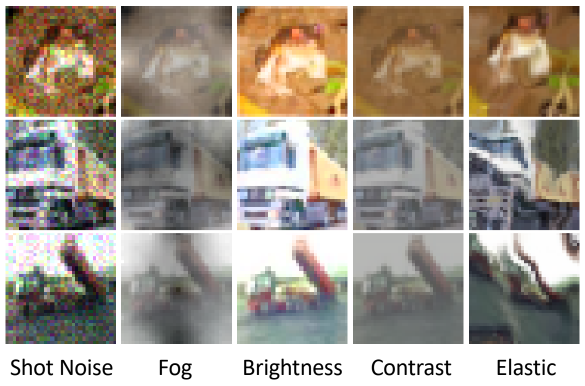


Figure 4.2: Examples of corrupted images. Images from CIFAR-10 are coruppted by the method of [6].

Such image corruption has been considered a data augmentation method, where the corrupted images are treated as ID samples. It is widely recognized [6] that CNNs trained only on clean images tend to fail to classify corrupted images correctly. Researchers have paid attention to generalizing the models to such image corruption [106–108]. It is then natural to use image corruption as a data augmentation method, requiring the created data to maintain the original samples' semantic contents.

To test the idea of using corrupted images as OOD samples, we conducted preliminary experiments. Concretely, we use the method to create the ImageNet-C dataset [6], which is publicly available by the authors[1] to synthesize various types of image corruption. It can synthesize 19 types of corruption, for each of which we can specify 5 severity levels. We choose 15 out of 19 corruption types that are originally assumed for training uses.

Choosing CIFAR-100 for ID samples, we train a network (i.e., Wide Resnet 40-4) in the following four settings. The first is to train the model using only the original, clean images. We apply the standard data augmentation (i.e., random crop and horizontal flip), which is also the case with the rest of the three. The second is train the model with the corrupted images, where image corruption is treated as data

[1]https://github.com/hendrycks/robustness

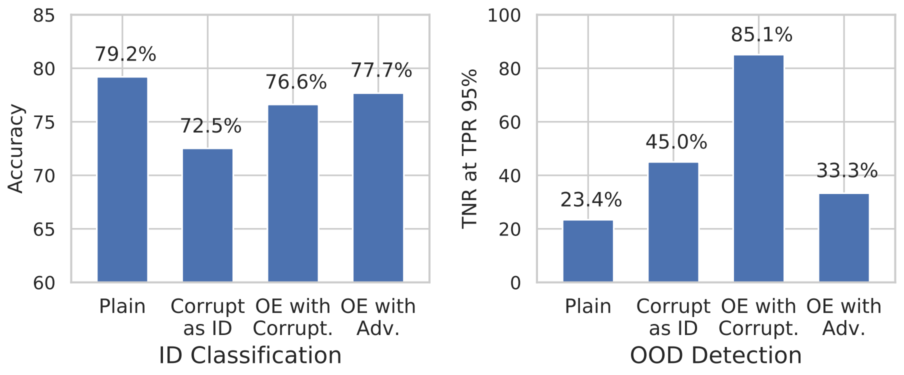


Figure 4.3: Comparison between classification and OOD detection performance of three models. "Plain" indicates the model trained with the standard training, while "Corrupt as ID" and "Corrupt as OOD" apply the ImageNet-C corruptions to the training dataset and utilize them as ID and OOD, respectively. OOD detection performance is evaluated by *TNR at TPR 95%*.

augmentation; in other words, the model is trained so as to classify the corrupted images to their original ground truth classes. The third is to train the model with 1:1 population of clean images and corrupted images, in which the clean images are treated as ID samples and the corrupted images are treated as OOD samples, and then the loss (4.1) is minimized as in Outlier Exposure (OE). The image corruption employed in the last two settings is randomly chosen from the 5 severity levels of the 15 corruptions. For the sake of comparison, we also consider another setting for OE, where we create adversarial examples from ID samples and use them as OOD samples, as in done in many studies on OOD detection [4, 89].

We tested the four models in terms of ID classification accuracy and OOD detection performance. We assumed the standard datasets (i.e., CIFAR-10, TIN, LSUN, iSUN, SVHN, and Food-101) for OOD, following previous studies; see Sec. 4.4.1 for details of the experiments. Figure 4.3 shows the results. It is observed that the two models trained with corrupted images lose the ID classification accuracy whereas they achieve better OOD detection performance.

These results have several implications. On the one hand, we can confirm some of the corrupted images do work as OOD samples as we intend, as the third model (i.e., OE using the corrupted images) improves OOD detection performance. This is also supported by the fact that the second model (i.e., the one trained using corrupted images as augmented data) loses ID classification accuracy. This performance deterioration is understandable because the second model uses all the corrupted images as ID samples, but some do not preserve the original semantic contents due to their severity; learning them as ID samples will harm the classification accuracy. On the other hand, some of the corrupted images work as ID samples; those with mild corruption levels preserve the image contents. This is verifiable by the fact that the third model loses ID classification accuracy (i.e., 79.2% $\rightarrow$ 76.6%). It is also noted that the last model using adversarial examples as OOD does not perform well on OOD detection compared with the third model using corrupted images as OOD samples.

Based on the above results and discussion, we pose the following conjectures:

- First, we should treat some of the corrupted images as ID samples and some of them as OOD samples. It will not be wise to treat all of them as either ID or OOD as we do in the above experiments.
- The employed image corruption will continuously cover the spectrum between ID and OOD samples due to the five severity levels of 15 corruptions.

These suggest that we could make further improvements by considering the *intermediate* region between ID and OOD. If we can assign intermediate labels to them, we could extend the OOD detection performance while maintaining the ID classification accuracy.

### 4.3.3 Soft Labels for Intermediate between ID and OOD

The question is how to assign soft labels to the intermediate samples between ID and OOD. Our solution is based on two ideas. One is to use a model $f(x)$ trained on the ID classification task using $\mathcal{D}_{\mathrm{in}}$ to obtain the soft labels. The other is to assign a soft label to a corrupted image $x' = T(x)$ according to the image corrupting transformation $T$, which is one of the predefined transformations. This is a natural extension of the above finding.

We denote an image corrupting transformation by $x' = T_i(x)$, where $i(= 1, \ldots, 75(= 15 \times 5))$ is an index indicating one of the 15 image corruption types with five severity levels. We apply $T_i$ to each sample of $\mathcal{D}_{\mathrm{in}}$. Let $\mathcal{D}^*_{\mathrm{in},i}$ be the set of the corrupted images.

Making the model $f(x)$ trained with clean images classify each sample of $\mathcal{D}^*_{\text{in},i}$, we calculate the classification accuracy $\text{acc}_i$ as

$$\text{acc}_i = \frac{1}{N}\sum_{n=1}^{N} 1(\hat{k}_n = k_n), \tag{4.2}$$

where $N = |\mathcal{D}_{\text{in}}| = |\mathcal{D}^*_{\text{in},i}|$; $\hat{k}_n$ and $k_n$ are the predicted and true class indexes, respectively. We use $\text{acc}_i$ for obtaining a soft label as explained below.

We train a new model $f^*$ using $\mathcal{D}_{\text{in}}$ and their corrupted images in the following way. Choosing a sample $x$ from $\mathcal{D}_{\text{in}}$, we first decide whether we apply image corruption to it according to a probability $\gamma$. If not, we use the original hard label $y$ and employ the loss $-\log f_k(x)$, where $k$ is the true class index. If yes, we choose and apply a transformation $T_i$ to $x$ and obtain a corrupted image $x' = T_i(x)$. We set the soft target label $t \in \mathbb{R}^K$ for $x'$ as follows:

$$t_j = \begin{cases} \text{acc}_i & \text{if } j = k, \\ \frac{1-\text{acc}_i}{K-1} & \text{otherwise}, \end{cases} \tag{4.3}$$

where $j = 1, \dots, K$ and $k$ is the true class index of $x$. This soft label is considered to be an interpolation between the hard label for the true class and the uniform label for OOD, as shown in Fig. 4.4.

When choosing $T_i$ for each $x \in \mathcal{D}_{\text{in}}$, we choose one so that the soft label $t$ distributes as uniformly in the label space as possible over $\mathcal{D}_{\text{in}}$. Rigorously, the maximum element of $t$ (i.e., $\text{argmax}_l\, t_l$), which is given by $\text{acc}_i$ as in (4.3), distributes uniformly in the range $[1/K, 1]$. To do this, we sample a random number $\alpha$ from a uniform distribution of the range $[1/K, 1]$ and search the nearest neighbor $\text{acc}_i$ to $\alpha$ from the pre-computed set $\{\text{acc}_i\}_{i=1,\dots,75}$ and choose the corresponding transformation $T_i$.

The procedure of creating a sample $(x, y)$ with or without image corruption from $\mathcal{D}_{\text{in}}$ is summarized in Algorithm 1; $(x, y)$ is either the pair of an original image and a hard label or the pair of a corrupted image and a soft label (i.e., $y_j \leftarrow t_j$ for $j = 1, \dots, K$). To train $f^*$, we minimize the cross entropy loss between a sample

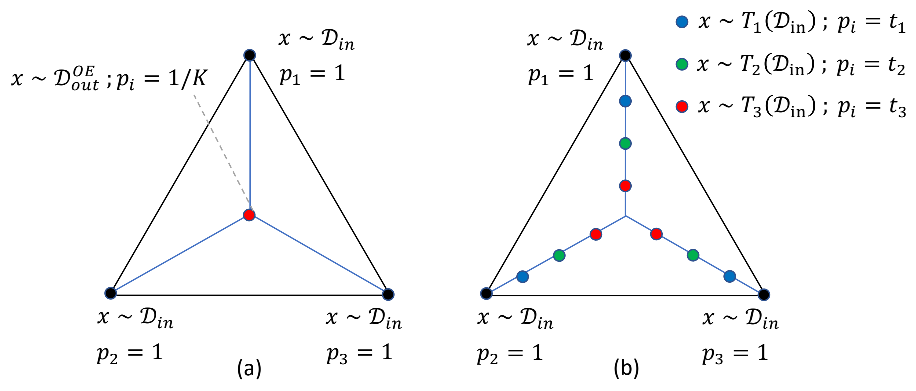

$x \sim \mathcal{D}_{out}^{OE}$ ; $p_i = 1/K$
$x \sim \mathcal{D}_{in}$
$p_1 = 1$
$x \sim \mathcal{D}_{in}$
$p_2 = 1$
$x \sim \mathcal{D}_{in}$
$p_3 = 1$
(a)
$x \sim \mathcal{D}_{in}$
$p_1 = 1$
● $x \sim T_1(\mathcal{D}_{\text{in}})$ ; $p_i = t_1$
● $x \sim T_2(\mathcal{D}_{\text{in}})$ ; $p_i = t_2$
● $x \sim T_3(\mathcal{D}_{\text{in}})$ ; $p_i = t_3$
$x \sim \mathcal{D}_{in}$
$p_2 = 1$
$x \sim \mathcal{D}_{in}$
$p_3 = 1$
(b)

Figure 4.4: Illustration of how target labels are assigned to ID and OOD samples in the case of three-class classification for (a) Outlier Exposure (OE) [7] and (b) our method. In OE, the target class probabilities for ID samples are 1 for the true class and 0 for others; those for OOD are all 1/3. In our method, there are intermediate samples lying between ID and OOD, which are created by applying image corrupting transformation $T_i$ to ID samples. Their target labels are set to constants determined for each $T_i$.

$(x, y)$ created as above and the prediction $\hat{y} = f^*(x)$:

$$\text{CE}(f^*(x), y) = -\sum_{j=1}^{K} y_j \log f_j^*(x). \tag{4.4}$$

**Algorithm 1:** Obtaining a training sample $(x, y)$

```
Input: Dataset $\mathcal{D}$, probability $\gamma$ of applying image corruption, and
  $\{\text{acc}_i\}_{i=1,\ldots,75}$
Output: An input image $x$ and its label $y$
Sample $(x, y)$ from $\mathcal{D}$;
$\beta \sim \text{Bernouli}(\gamma)$;
if $\beta = 0$ then
    $\alpha \sim \mathcal{U}(1/K, 1)$;
    $i \leftarrow \text{argmin}_j |\text{acc}_j - \alpha|$;
    $x \leftarrow T_i(x)$;
    Set $t$ according to (4.3);
    $y \leftarrow t$;
end
return $(x, y)$
```

There are two choices with how to train $f^*$. One is to initialize $f^* \leftarrow f$, where $f$ is the model trained using $\mathcal{D}_{\text{in}}$ on the ID class classification. The other is to train $f^*$

from scratch. We found in our experiments that the latter consistently works better and thus we will report its performance.

Previous studies employ the maximum softmax probability (MSP), i.e., $\max_l f_l(x)$, or its variant for detecting OOD samples [2, 3]. Our experiments show that the predicted entropy $-\sum_{j=1}^{K} f_j(x) \log f_j(x)$ shows slightly better performance consistently. Thus, we will report the results obtained using the entropy. Specifically, we classify an input providing the entropy higher than a threshold as OOD.

In our method, the probability $\gamma$ of applying image corrupting transformation to each sample is a hyperparameter. As will be shown later, the performance of the proposed method is not sensitive to its $\gamma$; it is stable in the range $[0.01, 0.4]$. We set $\gamma = 0.2$ in our experiments based on the ID classification performance on the validation data.

## 4.4 Experiments

We conduct experiments to evaluate the proposed method and compare it with existing methods.

### 4.4.1 Experimental Settings

We consider an image classification task, for which its training dataset $\mathcal{D}_{\text{in}}$ is given. Each OOD detection method first trains a network $f$ so that $f$ will accurately classify an ID input image $x$ as $\hat{y} = f(x)$. Then the method judges whether a new input $x$ is ID or OOD. We evaluate how accurately it can detect OOD samples.

**Datasets** In the experiments, we choose four ID datasets, i.e., CIFAR-10/100 [73], SVHN [66], and Food-101 [67]. We use one of the four datasets for ID and treat all others for OOD. Following Liang et al. [3], we also use Tiny ImageNet [64] (cropped and resized), LSUN [65] (cropped and resized), and iSUN [57] for OOD. CIFAR-10 and 100 [73] are datasets of 10 and 100 object categories, respectively, and each contains 50,000 and 10,000 samples for training and testing, respectively. SVHN [66] is a dataset of digit classification containing of 73,257 and 26,032 samples for training and testing. Food-101 [67] is a dataset of 101 food categories with 75,750 and 25,250 samples for training and testing. We resize the images of Food-101 to $32 \times 32$ pixels; the images of other datasets are of the same size.

**Networks and Training** We use DenseNet [72] and Wide Resnet [71]. More specifically, we use a 100-layer DenseNet with the bottleneck block and the growth rate of 12, denoted by DenseNet-100-12. We train it on an ID dataset for 300 epochs with mini-batch size of 64. For Wide Resnet, we use the 40-layer Wide Resnet with a widen factor of 4, denoted by WRN-40-4. We train it for 200 epochs with mini-batch size of 128. We employ weight regularization with factor 0.0001 for the former and with 0.0005 for the latter.

**Evaluation Metrics** Following previous studies [2–4, 25–27], we employ three standard metrics for evaluating OOD detection performance, i.e., *true negative rate at true positive rate 95% (TNR at TPR 95%)*, area under the ROC curve (AUROC), and area under the precision-recall curve (AUPR). Note that for all these metrics, a higher value indicates better performance.

### 4.4.2 Compared Methods

We compare our method with the following five methods. Following the recent studies [1, 5, 25–27, 74], we confine ourselves to the methods that do not need explicit OOD samples for training, if they are only a few. An exception is Outlier Exposure [7], which uses OOD samples for training but does not assume the similarity between them and the real OOD samples we encounter at test time.

**Baseline Method** [2] This method trains the network using $\mathcal{D}_{\text{in}}$ in the standard way. We call the resulting model the vanilla model. The method threshold the maximum softmax probability (MSP), also known as confidence, provided by the vanilla model for an input $x$ to judge if it is OOD. We use the predicted entropy from the same output in our experiments, because we found that it yields slightly better performance than MSP.

**Cosine Similarity** [5] This method uses a scaled cosine similarity instead of the dot product to yield logits in the final layer of the networks. Thus, it needs to change the layer design. It also employs a variant of input perturbation [3] for improved detection that is feasible without explicit OOD samples.

**Gram Matrices** [25] This method utilizes the Gram matrix calculated from the intermediate layer features of the vanilla model. It learns its statistics from $\mathcal{D}_{\text{in}}$ and use it to detect OOD inputs.

**MALCOM** [26] This method extends the Mahalanobis detector by the compression distance. The feature vectors are extracted both from the global average pooling (GAP) and the compression complexity pooling (CCP). All vectors are combined through the concatenation and are used to model the Mahalanobis distance.

**Outlier Exposure** [7] We have explained this method in Sec. 4.3.1. Following the study, we use 80 Million Tiny Images datasets [109] for $\mathcal{D}_{\text{out}}^{\text{OE}}$ in our experiments. We use the entropy instead of MSP due to the same reason as above.

### 4.4.3 Results

#### Out-of-Distribution Detection

We evaluate the OOD detection performance of the above methods for each ID dataset. Tables 4.1 and 4.2 show the performance measured by TNR at TPR 95% and AUROC, respectively. They show the mean and standard deviation over all the OOD datasets and over five trials of training. Those measured by AUPR and the detailed results showing the performance on each OOD dataset separately are given in Sec. 4.5.

It is seen that the proposed method achieves the best performance in almost all cases and in any evaluation metrics. We treat Outlier Exposure separately, as it uses external datasets; its performance should depend on their choice, although they are different from the true OOD datasets in our experiments. Nevertheless, our method performs comparably well and yields better results in several cases, i.e., when ID is CIFAR-100 and Food-101.

#### In-Distribution Classification

Some of the compared methods change either the network architecture or its training method [5,7,74] for OOD detection. These changes sometimes result in lower ID classification accuracy than the original network trained in the standard fashion.

Table 4.1: The OOD detection accuracy of the compared methods measured by TNR at TPR 95%. Outlier Exposure (OE) is treated separately, as its performance depends on the OOD dataset it assumes.

| Net | ID | Baseline | Cosine | Gram | MALCOM | Ours | OE |
|---|---|---|---|---|---|---|---|
| DenseNet | CIFAR-10 | 51.61(9.71) | 88.07(13.73) | 78.28(31.52) | 74.64(30.39) | **91.59(17.19)** | **93.83(8.65)** |
| | CIFAR-100 | 27.47(7.45) | 80.46(20.87) | 68.80(34.01) | 56.65(32.82) | **85.69(27.83)** | 51.81(15.93) |
| | SVHN | 68.53(4.40) | 74.27(10.70) | 93.39(6.69) | 98.70(1.50) | **99.32(0.82)** | **99.92(0.15)** |
| | Food-101 | 12.45(4.44) | 88.62(7.82) | 73.68(27.46) | 82.15(22.97) | **89.71(8.80)** | 63.46(28.98) |
| WRN | CIFAR-10 | 53.78(7.29) | 81.53(15.40) | 80.90(29.06) | 80.58(25.89) | **92.05(16.86)** | **95.28(6.51)** |
| | CIFAR-100 | 25.19(7.05) | 67.06(16.22) | 70.61(32.89) | 63.97(28.39) | **86.04(27.33)** | 42.82(17.18) |
| | SVHN | 73.35(3.27) | 75.99(11.31) | 94.00(6.35) | 98.42(1.66) | **99.44(0.69)** | **99.97(0.08)** |
| | Food-101 | 11.77(3.05) | 77.47(8.94) | 75.62(27.67) | 82.88(22.90) | **85.44(14.51)** | 80.02(17.49) |

Table 4.3 shows the ID classification accuracy for the standard model, the scaled cosine similarity, the proposed method, and Outlier Exposure. It is observed that the proposed method yields comparable performance to the standard model, whereas the cosine similarity underperforms slightly. All the other compared methods i.e., Gram Matrices [25] and MALCOM [26], use the standard model.

### Calibration Errors

It is well recognized [11] that modern neural networks tend to be over-confident with their prediction for classification tasks. Specifically, when they classify an input, the confidence of the prediction (i.e., the maximum softmax probability) tends to be larger than the expected prediction accuracy. It is said to be "calibrated" when the two are well aligned. How well a model is calibrated is evaluated by the expected calibration error (ECE). We calculate ECE for the model trained in each method. Table 4.4 shows the results. It is seen that the proposed method achieves the smallest ECE in most cases.

Table 4.2: The OOD detection accuracy of the compared methods measured by AUROC.

| Net | ID | Baseline | Cosine | Gram | MALCOM | Ours | OE |
|---|---|---|---|---|---|---|---|
| DenseNet | CIFAR-10 | 92.08(3.53) | 97.29(3.23) | 92.56(11.48) | 92.80(9.99) | **97.86(4.52)** | **98.54(1.69)** |
| | CIFAR-100 | 80.00(4.04) | 95.30(6.43) | 89.78(13.23) | 86.23(17.49) | **96.10(8.22)** | 88.91(4.53) |
| | SVHN | 92.05(2.15) | 94.06(2.76) | 98.44(1.56) | 99.57(0.40) | **99.79(0.21)** | **99.98(0.04)** |
| | Food-101 | 64.36(4.91) | **97.82(1.45)** | 91.66(10.16) | 95.46(5.90) | 97.00(2.69) | 87.43(10.78) |
| WRN | CIFAR-10 | 90.96(2.64) | 95.98(3.68) | 94.76(8.21) | 95.64(5.98) | **97.81(4.67)** | **98.36(1.06)** |
| | CIFAR-100 | 78.05(4.38) | 92.81(5.80) | 91.34(11.25) | 91.80(8.73) | **95.82(8.98)** | 89.51(4.43) |
| | SVHN | 93.06(1.29) | 94.56(2.76) | 98.61(1.43) | 99.58(0.41) | **99.78(0.16)** | **99.98(0.02)** |
| | Food-101 | 66.46(2.59) | **95.84(1.71)** | 92.54(9.41) | 95.53(6.16) | **95.85(4.24)** | 93.10(6.18) |

Table 4.3: The ID classification performance.

| | ID | Standard | Cosine | Ours | OE |
|---|---|---|---|---|---|
| DenseNet | CIFAR-10 | 95.13(0.09) | 94.89(0.13) | **95.37(0.12)** | 94.99(0.09) |
| | CIFAR-100 | 76.94(0.37) | 75.39(0.50) | **77.71(0.17)** | 75.93(0.33) |
| | SVHN | 96.36(0.10) | 95.98(0.18) | **96.63(0.04)** | 96.49(0.06) |
| | Food-101 | 42.42(0.23) | 40.53(0.24) | **42.88(0.36)** | **46.88(0.23)** |
| WRN | CIFAR-10 | 95.54(0.13) | 95.10(0.18) | **95.59(0.12)** | **95.79(0.10)** |
| | CIFAR-100 | **79.28(0.30)** | 76.66(0.30) | 79.21(0.11) | 76.58(0.23) |
| | SVHN | 96.67(0.03) | 96.40(0.10) | **96.83(0.06)** | 96.69(0.04) |
| | Food-101 | **44.92(0.15)** | 43.22(0.19) | 44.76(0.20) | **45.00(0.23)** |

### 4.4.4 Analyses

#### What Image Corrupting Transformation Is Good?

In the above experiments, we used 15 image corrupting transformations. The following questions will arise. Which corrupting transformations are more effective? How many transformations are necessary? To analyze these, we examine how the results will change depending on the number of transformations or a particular combination of selected corrupting transformation. We choose CIFAR-100 for the ID dataset and employ the Wide Resnet, which corresponds to a row in Table 4.1.

Table 4.4: Calibration errors of the models of the compared methods measured by the expected calibration error (ECE). A lower value is better.

| Net | ID | Standard | Cosine | Ours | OE |
|---|---|---|---|---|---|
| DenseNet | CIFAR-10 | 2.93(0.13) | 4.30(0.14) | **1.01(0.21)** | 2.07(0.15) |
| | CIFAR-100 | 12.12(0.44) | 20.48(0.61) | **3.63(0.44)** | 4.51(0.16) |
| | SVHN | 2.39(0.05) | 3.04(0.15) | **0.90(0.14)** | 1.00(0.09) |
| | Food-101 | 33.85(0.36) | 44.84(1.09) | **25.03(1.78)** | **10.39(0.49)** |
| WRN | CIFAR-10 | 2.86(0.13) | 4.14(0.10) | **0.58(0.08)** | 6.29(0.10) |
| | CIFAR-100 | 10.68(0.17) | 19.12(0.26) | **4.76(0.10)** | 17.24(0.28) |
| | SVHN | 2.18(0.07) | 2.93(0.11) | **1.56(0.55)** | 1.91(0.02) |
| | Food-101 | 28.34(0.18) | 47.59(0.54) | **19.26(1.41)** | 32.09(0.24) |

Table 4.5 shows OOD detection performance obtained for selected combinations of one to five corrupting transformations. The number(s) in $\{\cdot\}$ indicates the index of the 15 corruption types; see Sec. 4.6 for their details. Roughly speaking, using a combination of more corruption types tends to yield better performance. That said, some combinations of a few corruption types work much better than others. If we choose only a single corruption type, the best performer is *Elastic transform* (denoted by $\{12\}$ in the table); its performance (i.e., 77.88) is still better than other existing methods; see Table 4.1. Overall, we suggest to use an ensemble of all the corruption types, as it yields the best performance in the combinations we tested.

### Sensitivity to $\gamma$

The proposed method has a hyperparameter $\gamma$, which is the probability of applying image corruption to each sample at training time. We evaluate the sensitivity of the results to its choice. Figure 4.5 shows the ID classification accuracy and the OOD detection accuracy for different $\gamma$'s ranging in $[0, 1]$. We can observe the following. First, fortunately, the results are not sensitive to $\gamma$, especially for the range of $[0, 0.5]$. Second, they show similar tendencies; both decrease as $\gamma$ increases. From the result, it seems reasonable to choose it depending on the ID classification accuracy. Note that we cannot determine $\gamma$ based on the OOD detection accuracy, as it is not available without OOD samples. We chose $\gamma = 0.2$ based on this consideration in the above

Table 4.5: The OOD detection result in TNR at TPR 95% utilizing different set of the image-corrupting transformatoins in the training.

| Corruption | TNR |
|---|---|
| {0} | 37.20 |
| {1} | 39.19 |
| {2} | 25.87 |
| {3} | 34.94 |
| {4} | 37.20 |
| {5} | 37.88 |
| {6} | 32.79 |
| {7} | 26.89 |
| {8} | 34.90 |
| {9} | 30.44 |
| {10} | 30.23 |
| {11} | 26.75 |
| {12} | 77.88 |
| {13} | 31.22 |
| {14} | 62.98 |
| {0, 1, 2} | 59.70 |
| {3, 4, 5} | 48.65 |
| {6, 7, 8} | 76.84 |
| {9, 10, 11} | 64.98 |
| {12, 13, 14} | 58.44 |
| {0, 1, 2, 3, 4} | 78.80 |
| {5, 6, 7, 8, 9} | 63.79 |
| {10, 11, 12, 13, 14} | 83.06 |
| All | **86.04** |

experiments. The results do not change that much if we choose $\gamma = 0.1$ or 0.3, due to the above insensitivity.

## 4.5 Additional Result in AUPR

Table 4.6 shows the OOD detection accuracy measured by AUPR, which we omit from Sec. 4.4.3. The same observation holds as the evaluation by TNR at TPR 95% and by AUROC.

## 4.6 List of Image-Corrupting Transformations

We used ImageNet-C module [12] for the image-corrupting transformations. We used the code publicly available at `https://github.com/hendrycks/robustness`.

Table 4.6: The OOD detection accuracy of the compared methods measured by AUPR.

| Net | ID | Baseline | Cosine | Gram | MALCOM | Ours | OE |
|---|---|---|---|---|---|---|---|
| DenseNet | CIFAR-10 | 90.66(9.45) | 96.13(4.84) | 87.24(20.05) | 91.28(12.19) | **97.36(5.01)** | **98.35(1.82)** |
| | CIFAR-100 | 79.30(6.30) | 94.06(7.69) | 85.78(17.96) | 85.42(17.82) | **95.95(7.37)** | 87.97(3.78) |
| | SVHN | 94.55(3.95) | 96.71(2.40) | 96.56(3.69) | 99.83(0.14) | **99.89(0.12)** | **99.99(0.02)** |
| | Food-101 | 80.22(5.76) | **98.94(0.64)** | 95.03(6.06) | 97.90(2.71) | 98.36(1.47) | 93.69(4.98) |
| WRN | CIFAR-10 | 87.58(6.64) | 94.52(5.60) | 91.76(13.24) | 94.61(7.44) | **97.27(5.51)** | **98.52(1.08)** |
| | CIFAR-100 | 75.40(5.49) | 91.47(6.77) | 88.40(14.21) | 91.25(8.93) | **95.25(9.24)** | 90.52(3.16) |
| | SVHN | 95.44(1.58) | 97.15(1.73) | 97.08(3.30) | 99.83(0.16) | **99.90(0.08)** | **99.99(0.01)** |
| | Food-101 | 81.62(6.23) | 97.88(1.11) | 95.67(5.48) | **97.92(2.87)** | 97.77(2.18) | 96.25(3.35) |

Table 4.7: The OOD detection result in TNR at TPR 95% utilizing different set of the corruption(s) in the training.

| # | Corruption | # | Corruption |
|---|---|---|---|
| 0 | Gaussian noise | 8 | Frost |
| 1 | Shot noise | 9 | Fog |
| 2 | Impulse noise | 10 | Brightness |
| 3 | Defocus blur | 11 | Contrast |
| 4 | Frosted glass blur | 12 | Elastic transform |
| 5 | Motion blur | 13 | Pixelate |
| 6 | Zoom blur | 14 | JPEG compression |
| 7 | Snow | | |

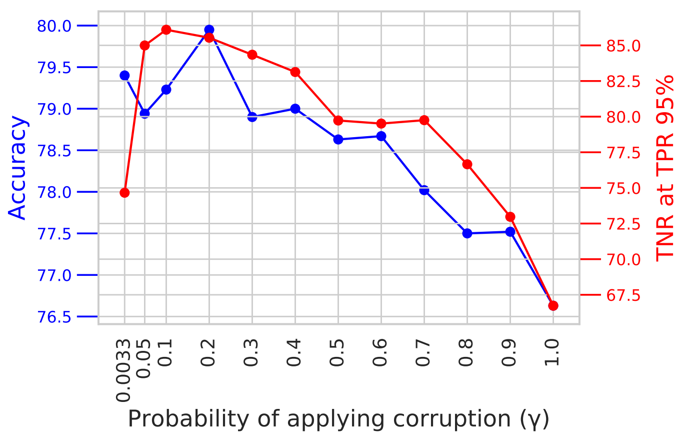


Figure 4.5: ID classification accuracy and OOD detection accuracy (TNR at TPR 95%) vs. the probability $\gamma$ of applying corruption to each image.

## 4.7 Results with Individual OOD Datasets

In Sec. 4.4.3, we report only the *average* of the detection scores for the multiple OOD datasets we employed in the experiments. Tables 4.8 - 4.13 report individual detection scores (i.e., TNR at TPR 95%, AUROC, and AUPR) for each OOD dataset. Each score is the average of five trials.

## 4.8 Conclusion

In this chapter, we have presented a new method for OOD detection. Questioning the premise of previous studies that ID and OOD samples are separated distinctly, we consider samples lying in the intermediate of the two and use them for training a network. We generate such samples using multiple image transformations that corrupt input images in various ways and with different severity levels. Specifically, applying one of the image transformations to ID samples, we assign the generated samples a soft target label representing how distant they are from the ID region. We compute the distance by using a network trained on clean ID samples for classifying those samples; we calculate their mean classification accuracy and use it to create the soft label. We then train the same network from scratch using the original ID samples and the generated samples with soft labels. The trained network can classify ID samples accurately. We detect OOD samples by thresholding the entropy of the predicted softmax probability. The experimental results show that our method outperforms the previous state-of-the-art in the standard benchmark tests widely employed in previous studies. We have also analyzed the effect of the number and particular combinations of image corrupting transformations on the performance.

Table 4.8: The OOD detection performance in TNR at TPR 95%. The network architecture is DenseNet-100-12.

| ID | OOD | Baseline | Cosine | Gram | MALCOM | Ours | OE |
|---|---|---|---|---|---|---|---|
| CIFAR-10 | CIFAR-100 | 41.22(1.10) | 60.59(1.20) | 26.90(0.26) | 24.86(0.40) | 47.23(0.47) | 71.55(0.72) |
| | Food-101 | 47.96(1.93) | 69.06(4.28) | 21.13(0.67) | 20.33(1.19) | 88.50(3.18) | 93.67(0.82) |
| | iSUN | 59.66(4.76) | 96.08(1.19) | 98.75(0.26) | 94.12(0.57) | 99.71(0.13) | 98.63(0.42) |
| | LSUNc | 55.13(2.09) | 95.74(0.53) | 88.82(0.72) | 81.39(2.07) | 99.27(0.20) | 98.65(0.40) |
| | LSUNr | 63.64(3.66) | 96.18(1.52) | 99.32(0.12) | 95.36(0.68) | 99.76(0.15) | 98.85(0.33) |
| | TINc | 56.01(3.91) | 94.96(0.35) | 96.65(0.47) | 91.40(1.00) | 99.45(0.19) | 96.86(1.12) |
| | TINr | 54.20(5.27) | 94.17(0.56) | 98.41(0.28) | 94.71(0.61) | 99.44(0.21) | 95.12(1.67) |
| | SVHN | 35.04(4.77) | 97.76(0.22) | 96.27(0.46) | 94.98(0.80) | 99.37(0.10) | 97.31(1.44) |
| CIFAR-100 | CIFAR-10 | 20.22(0.74) | 29.96(2.38) | 10.55(0.43) | 1.34(0.08) | 12.88(0.81) | 17.57(0.74) |
| | Food-101 | 39.42(2.40) | 68.01(4.59) | 14.29(1.31) | 3.70(0.33) | 85.96(3.52) | 49.98(2.37) |
| | iSUN | 25.85(6.36) | 92.87(0.45) | 96.23(0.63) | 82.96(1.81) | 97.60(0.65) | 59.53(3.48) |
| | LSUNc | 28.16(1.88) | 84.61(2.56) | 67.16(1.30) | 53.05(1.25) | 96.38(1.02) | 72.87(1.49) |
| | LSUNr | 29.07(7.10) | 93.39(0.75) | 97.59(0.39) | 85.12(2.06) | 98.65(0.70) | 61.90(3.36) |
| | TINc | 30.11(5.32) | 93.28(0.84) | 90.65(0.97) | 74.45(1.79) | 97.95(0.92) | 52.03(4.22) |
| | TINr | 27.06(6.31) | 93.68(0.88) | 95.96(0.39) | 84.26(1.43) | 98.22(0.96) | 46.25(4.88) |
| | SVHN | 19.85(3.09) | 87.86(3.19) | 77.94(1.95) | 68.30(3.35) | 97.87(0.28) | 54.35(11.76) |
| SVHN | CIFAR-10 | 65.90(2.55) | 69.44(8.53) | 81.82(0.58) | 96.68(0.42) | 98.47(0.47) | 99.96(0.03) |
| | CIFAR-100 | 64.67(2.28) | 67.73(8.77) | 83.96(0.91) | 97.73(0.42) | 97.77(0.42) | 99.89(0.02) |
| | Food-101 | 64.32(5.90) | 68.66(13.45) | 91.87(1.33) | 99.38(0.08) | 99.18(0.30) | 99.99(0.00) |
| | iSUN | 70.55(1.66) | 78.05(8.73) | 99.28(0.29) | 99.99(0.01) | 100.00(0.00) | 100.00(0.00) |
| | LSUNc | 68.18(4.14) | 79.67(7.43) | 93.27(0.88) | 96.20(0.47) | 99.22(0.16) | 99.54(0.07) |
| | LSUNr | 69.71(1.82) | 74.38(8.83) | 99.47(0.20) | 99.99(0.01) | 100.00(0.00) | 100.00(0.00) |
| | TINc | 72.40(1.97) | 78.90(10.00) | 98.34(0.54) | 99.67(0.06) | 99.96(0.01) | 99.98(0.00) |
| | TINr | 72.51(2.40) | 77.33(10.19) | 99.11(0.34) | 99.95(0.03) | 99.99(0.01) | 100.00(0.00) |
| Food-101 | CIFAR-10 | 13.81(1.58) | 76.34(3.56) | 22.37(0.88) | 36.37(0.64) | 84.03(6.59) | 99.92(0.01) |
| | CIFAR-100 | 12.30(1.09) | 76.07(3.67) | 31.57(0.54) | 50.41(0.37) | 86.45(5.58) | 99.75(0.04) |
| | iSUN | 10.67(3.51) | 94.04(2.84) | 93.20(1.16) | 98.33(0.32) | 96.86(1.48) | 39.73(10.71) |
| | LSUNc | 20.34(2.80) | 93.41(1.59) | 79.20(0.96) | 86.14(0.71) | 74.80(2.40) | 64.09(1.83) |
| | LSUNr | 11.76(4.43) | 92.80(3.30) | 94.10(1.30) | 98.39(0.36) | 98.22(0.84) | 35.97(10.52) |
| | TINc | 10.88(2.59) | 91.73(2.40) | 85.96(1.71) | 93.09(0.43) | 85.18(4.05) | 40.75(7.04) |
| | TINr | 8.24(2.83) | 90.59(2.91) | 90.89(1.31) | 96.12(0.22) | 92.71(2.64) | 33.13(9.00) |
| | SVHN | 11.56(3.31) | 94.00(2.63) | 92.16(1.44) | 98.30(0.50) | 99.46(0.13) | 94.37(1.95) |

Table 4.9: The OOD detection performance in TNR at TPR 95%. The network architecture is WRN-40-4.

| ID | OOD | Baseline | Cosine | Gram | MALCOM | Ours | OE |
|---|---|---|---|---|---|---|---|
| CIFAR-10 | CIFAR-100 | 42.85(1.03) | 56.42(2.64) | 31.97(0.69) | 35.05(0.89) | 48.31(1.64) | 78.48(0.47) |
| | Food-101 | 45.64(1.20) | 58.92(6.52) | 29.53(1.65) | 37.07(4.97) | 89.94(1.54) | 97.16(0.22) |
| | iSUN | 55.33(2.18) | 89.82(5.75) | 99.51(0.06) | 97.68(0.35) | 99.83(0.11) | 98.21(0.47) |
| | LSUNc | 61.08(1.71) | 92.76(2.29) | 92.14(0.57) | 90.26(0.98) | 99.60(0.09) | 99.12(0.20) |
| | LSUNr | 59.73(1.81) | 91.81(4.17) | 99.62(0.05) | 98.33(0.38) | 99.88(0.10) | 98.46(0.50) |
| | TINc | 53.83(1.94) | 84.87(8.26) | 97.81(0.15) | 95.44(0.43) | 99.73(0.12) | 96.85(0.62) |
| | TINr | 48.70(2.22) | 82.53(8.69) | 99.07(0.09) | 96.47(0.35) | 99.65(0.14) | 94.85(1.00) |
| | SVHN | 63.07(3.73) | 95.12(2.89) | 97.57(0.26) | 94.37(1.44) | 99.45(0.08) | 99.16(0.23) |
| CIFAR-100 | CIFAR-10 | 22.42(0.71) | 27.65(1.31) | 11.74(0.74) | 9.06(1.16) | 14.37(0.97) | 18.02(0.67) |
| | Food-101 | 42.92(0.98) | 66.70(1.27) | 19.92(1.07) | 23.30(6.37) | 87.40(2.20) | 47.73(1.95) |
| | iSUN | 19.97(1.94) | 71.00(6.47) | 96.42(0.11) | 81.17(5.21) | 97.25(0.96) | 39.35(6.75) |
| | LSUNc | 22.26(1.79) | 72.45(2.48) | 69.24(1.25) | 71.82(0.89) | 97.37(1.05) | 68.10(0.79) |
| | LSUNr | 22.86(2.24) | 70.31(6.09) | 97.69(0.12) | 81.24(6.84) | 98.44(0.65) | 43.89(6.90) |
| | TINc | 25.68(1.19) | 77.90(5.80) | 91.20(0.22) | 78.51(4.40) | 98.46(0.76) | 31.98(3.76) |
| | TINr | 23.41(1.57) | 74.97(6.40) | 95.97(0.39) | 82.81(4.09) | 98.35(0.59) | 27.32(4.71) |
| | SVHN | 22.02(1.98) | 75.52(9.10) | 82.66(0.53) | 83.88(2.80) | 96.68(0.35) | 66.13(5.96) |
| SVHN | CIFAR-10 | 72.46(1.17) | 71.63(10.81) | 84.12(1.70) | 96.23(0.40) | 99.03(0.10) | 99.99(0.00) |
| | CIFAR-100 | 72.52(1.51) | 69.07(10.59) | 84.71(1.55) | 96.28(0.30) | 98.07(0.29) | 99.98(0.01) |
| | Food-101 | 76.25(1.57) | 74.39(10.54) | 91.08(1.70) | 98.74(0.05) | 99.57(0.07) | 100.00(0.00) |
| | iSUN | 72.08(3.97) | 78.10(10.64) | 99.78(0.06) | 99.99(0.00) | 100.00(0.00) | 100.00(0.00) |
| | LSUNc | 71.37(2.37) | 81.98(6.99) | 94.07(0.31) | 96.56(0.37) | 98.94(0.51) | 99.77(0.04) |
| | LSUNr | 71.57(3.65) | 74.16(12.45) | 99.87(0.04) | 99.99(0.00) | 100.00(0.00) | 100.00(0.00) |
| | TINc | 75.71(2.93) | 80.44(10.55) | 98.80(0.17) | 99.65(0.09) | 99.96(0.02) | 100.00(0.00) |
| | TINr | 74.86(3.12) | 78.12(10.79) | 99.57(0.12) | 99.93(0.03) | 99.97(0.02) | 100.00(0.00) |
| Food-101 | CIFAR-10 | 13.62(0.24) | 67.86(2.41) | 24.16(0.63) | 37.98(1.27) | 64.90(11.29) | 99.99(0.01) |
| | CIFAR-100 | 12.74(0.53) | 65.73(1.83) | 33.27(0.76) | 49.83(1.02) | 68.25(10.00) | 100.00(0.00) |
| | iSUN | 10.90(3.16) | 80.31(8.28) | 94.57(0.97) | 98.15(0.70) | 98.00(0.59) | 78.38(6.39) |
| | LSUNc | 14.87(2.19) | 88.20(0.98) | 79.27(0.56) | 89.23(0.60) | 73.23(3.02) | 59.19(2.93) |
| | LSUNr | 11.49(2.99) | 78.11(8.84) | 95.26(0.98) | 98.38(0.62) | 98.40(0.42) | 76.63(8.16) |
| | TINc | 12.16(2.17) | 81.79(5.08) | 88.52(1.09) | 94.18(0.86) | 88.21(2.04) | 59.22(6.04) |
| | TINr | 10.57(2.43) | 77.87(7.15) | 92.88(0.92) | 96.19(0.84) | 94.51(1.23) | 67.18(8.29) |
| | SVHN | 7.80(2.72) | 79.85(5.21) | 96.99(0.80) | 99.12(0.32) | 98.05(0.68) | 99.57(0.17) |

Table 4.10: The OOD detection performance in AUROC. The network architecture is DenseNet-100-12.

| ID | OOD | Baseline | Cosine | Gram | MALCOM | Ours | OE |
|---|---|---|---|---|---|---|---|
| CIFAR-10 | CIFAR-100 | 89.48(0.15) | 90.21(0.22) | 72.63(0.43) | 71.15(0.30) | 86.06(0.42) | 94.19(0.13) |
| | Food-101 | 91.75(0.48) | 93.74(1.06) | 72.79(0.54) | 81.03(1.08) | 97.68(0.65) | 98.57(0.09) |
| | iSUN | 94.44(0.78) | 99.10(0.25) | 99.73(0.05) | 98.78(0.10) | 99.88(0.03) | 99.45(0.13) |
| | LSUNc | 93.52(0.50) | 99.09(0.12) | 97.46(0.09) | 96.46(0.41) | 99.82(0.04) | 99.56(0.07) |
| | LSUNr | 95.10(0.53) | 99.09(0.31) | 99.84(0.03) | 98.92(0.12) | 99.89(0.04) | 99.50(0.11) |
| | TINc | 93.66(0.87) | 98.87(0.08) | 99.23(0.08) | 98.30(0.17) | 99.85(0.05) | 99.10(0.23) |
| | TINr | 93.30(1.06) | 98.72(0.12) | 99.63(0.06) | 98.88(0.11) | 99.84(0.05) | 98.79(0.32) |
| | SVHN | 85.36(4.85) | 99.47(0.07) | 99.15(0.09) | 98.86(0.16) | 99.83(0.03) | 99.22(0.30) |
| CIFAR-100 | CIFAR-10 | 77.86(0.28) | 78.68(1.04) | 63.93(0.47) | 45.32(0.32) | 74.46(0.46) | 78.98(0.45) |
| | Food-101 | 86.97(0.69) | 94.76(0.64) | 70.86(0.95) | 71.24(1.52) | 97.27(0.65) | 91.36(0.53) |
| | iSUN | 80.25(1.61) | 98.48(0.09) | 99.14(0.10) | 97.04(0.32) | 99.47(0.15) | 90.21(1.01) |
| | LSUNc | 80.44(0.84) | 96.89(0.56) | 92.43(0.25) | 91.54(0.49) | 99.23(0.18) | 94.02(0.40) |
| | LSUNr | 81.99(1.55) | 98.64(0.15) | 99.45(0.06) | 97.25(0.36) | 99.69(0.13) | 90.98(1.19) |
| | TINc | 81.94(1.53) | 98.60(0.20) | 97.97(0.14) | 95.61(0.37) | 99.55(0.17) | 88.04(1.34) |
| | TINr | 80.87(2.17) | 98.66(0.20) | 99.10(0.05) | 97.18(0.30) | 99.62(0.18) | 86.17(1.64) |
| | SVHN | 77.28(2.12) | 97.70(0.49) | 95.39(0.38) | 94.68(0.39) | 99.46(0.05) | 91.52(2.30) |
| SVHN | CIFAR-10 | 91.55(1.35) | 93.14(1.93) | 95.61(0.18) | 98.99(0.06) | 99.57(0.10) | 99.99(0.00) |
| | CIFAR-100 | 90.95(1.18) | 92.60(2.16) | 96.38(0.19) | 99.17(0.06) | 99.39(0.09) | 99.97(0.00) |
| | Food-101 | 90.33(3.62) | 92.58(3.88) | 98.03(0.35) | 99.63(0.02) | 99.71(0.06) | 99.99(0.00) |
| | iSUN | 92.88(0.93) | 95.04(2.19) | 99.80(0.08) | 99.96(0.01) | 99.97(0.02) | 100.00(0.00) |
| | LSUNc | 91.23(2.38) | 94.81(2.33) | 98.57(0.15) | 99.07(0.09) | 99.77(0.05) | 99.88(0.02) |
| | LSUNr | 92.70(1.16) | 94.14(2.33) | 99.84(0.05) | 99.96(0.01) | 99.97(0.02) | 100.00(0.00) |
| | TINc | 93.32(1.13) | 95.32(2.52) | 99.55(0.11) | 99.85(0.02) | 99.96(0.02) | 99.99(0.00) |
| | TINr | 93.43(1.35) | 94.86(2.59) | 99.76(0.08) | 99.93(0.01) | 99.97(0.02) | 100.00(0.00) |
| Food-101 | CIFAR-10 | 67.71(1.13) | 95.57(0.62) | 71.75(0.70) | 83.55(0.54) | 95.56(1.95) | 99.98(0.00) |
| | CIFAR-100 | 65.13(0.82) | 95.46(0.67) | 76.99(0.23) | 87.41(0.33) | 96.07(1.69) | 99.92(0.02) |
| | iSUN | 62.06(4.88) | 98.83(0.51) | 98.53(0.25) | 99.49(0.08) | 99.18(0.34) | 79.95(7.14) |
| | LSUNc | 70.21(1.73) | 98.73(0.29) | 94.36(0.28) | 96.92(0.14) | 92.10(0.91) | 88.06(0.67) |
| | LSUNr | 63.85(4.67) | 98.62(0.58) | 98.60(0.26) | 99.47(0.09) | 99.54(0.19) | 79.00(6.89) |
| | TINc | 61.37(3.28) | 98.38(0.49) | 96.87(0.38) | 98.48(0.10) | 95.72(1.04) | 78.89(4.43) |
| | TINr | 58.90(4.24) | 98.18(0.55) | 97.87(0.30) | 99.09(0.07) | 97.92(0.62) | 75.42(6.11) |
| | SVHN | 65.62(4.69) | 98.82(0.44) | 98.33(0.21) | 99.30(0.10) | 99.86(0.04) | 98.20(0.58) |

Table 4.11: The OOD detection performance in AUROC. The network architecture is WRN-40-4.

| ID | OOD | Baseline | Cosine | Gram | MALCOM | Ours | OE |
|---|---|---|---|---|---|---|---|
| CIFAR-10 | CIFAR-100 | 86.54(0.13) | 88.87(0.81) | 79.50(0.24) | 83.03(0.68) | 85.56(0.29) | 95.65(0.07) |
| | Food-101 | 88.16(0.40) | 91.54(1.71) | 81.71(0.66) | 88.15(1.29) | 98.00(0.27) | 98.85(0.05) |
| | iSUN | 91.99(0.60) | 98.02(1.00) | 99.85(0.01) | 99.35(0.08) | 99.84(0.06) | 98.75(0.12) |
| | LSUNc | 93.43(0.41) | 98.57(0.43) | 98.39(0.09) | 98.17(0.18) | 99.83(0.05) | 99.11(0.04) |
| | LSUNr | 93.26(0.53) | 98.39(0.72) | 99.89(0.01) | 99.41(0.09) | 99.85(0.07) | 98.77(0.12) |
| | TINc | 90.98(0.51) | 97.06(1.60) | 99.50(0.03) | 98.97(0.09) | 99.84(0.05) | 98.64(0.10) |
| | TINr | 89.22(1.06) | 96.47(1.76) | 99.77(0.02) | 99.18(0.06) | 99.80(0.07) | 98.21(0.15) |
| | SVHN | 94.09(0.73) | 98.94(0.72) | 99.47(0.06) | 98.88(0.27) | 99.81(0.03) | 98.91(0.26) |
| CIFAR-100 | CIFAR-10 | 78.82(0.43) | 77.84(0.85) | 67.34(0.21) | 70.57(1.11) | 72.16(1.33) | 81.89(0.12) |
| | Food-101 | 88.24(0.37) | 94.41(0.15) | 78.05(0.46) | 86.33(1.95) | 97.50(0.38) | 92.46(0.19) |
| | iSUN | 74.04(1.22) | 94.52(1.29) | 99.10(0.03) | 96.40(0.88) | 99.41(0.15) | 89.53(1.87) |
| | LSUNc | 78.34(0.80) | 94.46(0.68) | 93.39(0.20) | 94.97(0.25) | 99.41(0.18) | 94.59(0.18) |
| | LSUNr | 75.45(1.03) | 94.48(1.27) | 99.39(0.02) | 96.44(1.12) | 99.63(0.11) | 90.83(1.56) |
| | TINc | 76.43(1.11) | 95.92(1.07) | 98.09(0.08) | 96.11(0.68) | 99.64(0.13) | 86.90(1.64) |
| | TINr | 74.20(1.23) | 95.26(1.31) | 99.05(0.07) | 96.66(0.71) | 99.60(0.10) | 85.34(1.95) |
| | SVHN | 78.87(1.30) | 95.62(1.58) | 96.34(0.17) | 96.94(0.53) | 99.21(0.08) | 94.52(0.76) |
| SVHN | CIFAR-10 | 92.56(0.48) | 93.62(2.62) | 96.34(0.41) | 99.01(0.09) | 99.67(0.05) | 99.99(0.00) |
| | CIFAR-100 | 92.49(0.59) | 92.61(2.68) | 96.59(0.37) | 99.04(0.07) | 99.45(0.06) | 99.99(0.00) |
| | Food-101 | 94.04(0.62) | 94.40(2.14) | 97.88(0.37) | 99.56(0.04) | 99.82(0.02) | 99.99(0.00) |
| | iSUN | 92.92(1.43) | 95.14(2.69) | 99.92(0.02) | 99.98(0.01) | 99.92(0.03) | 99.99(0.00) |
| | LSUNc | 92.12(1.00) | 95.65(1.76) | 98.73(0.02) | 99.20(0.07) | 99.69(0.10) | 99.93(0.00) |
| | LSUNr | 92.44(1.51) | 94.08(3.47) | 99.94(0.01) | 99.99(0.00) | 99.90(0.03) | 99.99(0.00) |
| | TINc | 94.09(1.15) | 95.76(2.44) | 99.66(0.05) | 99.88(0.02) | 99.92(0.02) | 99.99(0.00) |
| | TINr | 93.83(1.08) | 95.22(2.42) | 99.85(0.03) | 99.95(0.01) | 99.90(0.03) | 99.99(0.00) |
| Food-101 | CIFAR-10 | 68.62(0.13) | 94.06(0.50) | 74.00(0.44) | 83.32(0.64) | 90.23(3.54) | 100.00(0.00) |
| | CIFAR-100 | 67.00(0.32) | 93.40(0.47) | 79.18(0.51) | 86.75(0.52) | 90.90(3.12) | 100.00(0.00) |
| | iSUN | 66.28(2.72) | 96.53(1.45) | 98.81(0.22) | 99.53(0.15) | 99.49(0.14) | 92.90(2.27) |
| | LSUNc | 68.23(2.91) | 97.78(0.20) | 94.44(0.29) | 97.63(0.15) | 91.92(0.93) | 85.89(0.98) |
| | LSUNr | 66.29(2.41) | 96.13(1.61) | 98.90(0.20) | 99.52(0.16) | 99.61(0.11) | 92.35(2.73) |
| | TINc | 66.55(1.49) | 96.64(0.86) | 97.44(0.27) | 98.72(0.16) | 96.68(0.63) | 85.36(2.59) |
| | TINr | 65.36(1.84) | 95.92(1.31) | 98.32(0.23) | 99.10(0.19) | 98.51(0.37) | 88.41(3.10) |
| | SVHN | 63.36(2.73) | 96.25(1.15) | 99.23(0.17) | 99.68(0.10) | 99.49(0.17) | 99.87(0.07) |

Table 4.12: The OOD detection performance in AUPR. The network architecture is DenseNet-100-12.

| ID | OOD | Baseline | Cosine | Gram | MALCOM | Ours | OE |
|---|---|---|---|---|---|---|---|
| CIFAR-10 | CIFAR-100 | 90.64(0.11) | 88.44(0.44) | 62.57(0.68) | 69.23(0.37) | 84.77(0.50) | 93.96(0.18) |
| | Food-101 | 87.50(0.80) | 87.36(2.25) | 44.41(1.04) | 71.27(1.66) | 95.24(1.35) | 97.37(0.21) |
| | iSUN | 95.95(0.55) | 99.15(0.24) | 99.72(0.05) | 99.01(0.08) | 99.90(0.02) | 99.58(0.10) |
| | LSUNc | 94.79(0.44) | 98.99(0.14) | 96.21(0.20) | 96.74(0.39) | 99.81(0.04) | 99.60(0.07) |
| | LSUNr | 96.10(0.41) | 99.05(0.32) | 99.81(0.02) | 99.05(0.10) | 99.90(0.03) | 99.58(0.09) |
| | TINc | 94.92(0.75) | 98.81(0.08) | 98.85(0.10) | 98.42(0.17) | 99.85(0.05) | 99.21(0.19) |
| | TINr | 94.61(0.85) | 98.62(0.10) | 99.46(0.14) | 98.94(0.10) | 99.84(0.05) | 98.91(0.29) |
| | SVHN | 70.78(14.06) | 98.65(0.11) | 96.86(0.20) | 97.56(0.28) | 99.55(0.06) | 98.60(0.43) |
| CIFAR-100 | CIFAR-10 | 80.41(0.29) | 74.76(1.33) | 60.99(0.50) | 49.47(0.25) | 76.94(0.44) | 81.96(0.31) |
| | Food-101 | 80.20(0.83) | 91.52(0.90) | 50.39(1.22) | 61.04(2.36) | 94.65(1.22) | 86.62(0.77) |
| | iSUN | 83.84(1.22) | 98.57(0.07) | 99.05(0.11) | 97.68(0.24) | 99.49(0.15) | 90.87(0.93) |
| | LSUNc | 82.32(0.75) | 96.66(0.64) | 91.15(0.29) | 92.68(0.53) | 99.21(0.18) | 93.82(0.47) |
| | LSUNr | 84.11(1.18) | 98.66(0.14) | 99.38(0.08) | 97.68(0.30) | 99.69(0.15) | 90.87(1.34) |
| | TINc | 83.86(1.55) | 98.57(0.18) | 97.54(0.17) | 96.12(0.35) | 99.54(0.17) | 87.86(1.16) |
| | TINr | 83.08(1.97) | 98.62(0.16) | 98.92(0.05) | 97.46(0.28) | 99.60(0.20) | 86.05(1.28) |
| | SVHN | 66.54(3.26) | 95.13(0.82) | 88.85(1.00) | 91.21(0.54) | 98.45(0.15) | 85.74(3.22) |
| SVHN | CIFAR-10 | 95.28(1.25) | 96.84(0.88) | 89.36(0.49) | 99.66(0.02) | 99.82(0.05) | 99.99(0.00) |
| | CIFAR-100 | 94.69(1.19) | 96.48(1.06) | 91.46(0.41) | 99.72(0.02) | 99.71(0.06) | 99.99(0.00) |
| | Food-101 | 87.14(6.69) | 92.51(3.88) | 97.62(0.46) | 99.72(0.02) | 99.74(0.06) | 100.00(0.00) |
| | iSUN | 96.43(0.74) | 97.90(1.00) | 99.31(0.26) | 99.99(0.00) | 99.99(0.00) | 100.00(0.00) |
| | LSUNc | 94.40(2.42) | 97.30(1.37) | 97.09(0.31) | 99.67(0.03) | 99.90(0.02) | 99.95(0.01) |
| | LSUNr | 95.95(0.99) | 97.22(1.21) | 99.48(0.18) | 99.99(0.00) | 99.99(0.01) | 100.00(0.00) |
| | TINc | 96.20(1.11) | 97.82(1.24) | 98.85(0.30) | 99.95(0.00) | 99.99(0.01) | 100.00(0.00) |
| | TINr | 96.29(1.24) | 97.59(1.27) | 99.30(0.24) | 99.98(0.00) | 99.99(0.01) | 100.00(0.00) |
| Food-101 | CIFAR-10 | 84.13(0.53) | 98.03(0.27) | 83.10(0.51) | 92.46(0.31) | 97.67(1.03) | 99.99(0.00) |
| | CIFAR-100 | 82.47(0.35) | 97.97(0.28) | 86.38(0.27) | 94.18(0.19) | 97.89(0.90) | 99.95(0.01) |
| | iSUN | 82.43(2.65) | 99.53(0.20) | 99.37(0.11) | 99.83(0.02) | 99.62(0.15) | 91.30(3.31) |
| | LSUNc | 84.66(0.86) | 99.41(0.13) | 96.58(0.21) | 98.56(0.08) | 95.60(0.53) | 93.47(0.34) |
| | LSUNr | 82.01(2.50) | 99.39(0.25) | 99.33(0.12) | 99.80(0.03) | 99.77(0.09) | 90.17(3.35) |
| | TINc | 80.06(1.82) | 99.26(0.23) | 98.36(0.21) | 99.35(0.05) | 97.67(0.53) | 89.33(2.18) |
| | TINr | 78.94(2.38) | 99.17(0.24) | 98.89(0.17) | 99.62(0.03) | 98.86(0.29) | 87.98(2.97) |
| | SVHN | 67.02(4.30) | 98.74(0.43) | 98.22(0.23) | 99.41(0.09) | 99.82(0.05) | 97.34(0.80) |

Table 4.13: The OOD detection performance in AUPR. The network architecture is WRN-40-4.

| ID | OOD | Baseline | Cosine | Gram | MALCOM | Ours | OE |
|---|---|---|---|---|---|---|---|
| CIFAR-10 | CIFAR-100 | 83.29(0.16) | 86.81(0.94) | 74.26(0.45) | 82.88(0.80) | 83.03(0.39) | 95.81(0.08) |
| | Food-101 | 72.55(1.34) | 84.01(2.99) | 64.32(1.61) | 80.89(2.02) | 96.38(0.65) | 98.39(0.05) |
| | iSUN | 92.56(0.79) | 98.20(0.89) | 99.85(0.01) | 99.48(0.06) | 99.88(0.05) | 99.20(0.07) |
| | LSUNc | 93.19(0.52) | 98.43(0.46) | 98.19(0.11) | 98.38(0.15) | 99.84(0.05) | 99.37(0.03) |
| | LSUNr | 93.26(0.72) | 98.40(0.68) | 99.88(0.01) | 99.50(0.07) | 99.87(0.06) | 99.18(0.09) |
| | TINc | 90.08(0.85) | 96.98(1.62) | 99.44(0.04) | 99.08(0.07) | 99.86(0.05) | 99.00(0.08) |
| | TINr | 87.95(1.76) | 96.31(1.76) | 99.71(0.05) | 99.24(0.05) | 99.82(0.07) | 98.54(0.12) |
| | SVHN | 87.76(2.46) | 97.03(2.03) | 98.45(0.16) | 97.44(0.57) | 99.54(0.07) | 98.69(0.20) |
| CIFAR-100 | CIFAR-10 | 80.00(0.75) | 74.53(1.46) | 65.28(0.29) | 71.71(1.11) | 71.17(1.74) | 85.51(0.07) |
| | Food-101 | 80.92(0.69) | 90.60(0.29) | 63.42(0.81) | 81.47(2.34) | 95.11(0.73) | 89.94(0.17) |
| | iSUN | 75.65(1.40) | 95.04(1.06) | 99.00(0.04) | 97.05(0.68) | 99.48(0.14) | 92.24(1.47) |
| | LSUNc | 79.43(1.01) | 94.23(0.81) | 92.49(0.19) | 95.54(0.29) | 99.40(0.17) | 95.47(0.17) |
| | LSUNr | 75.08(0.79) | 94.64(1.17) | 99.36(0.02) | 96.82(0.96) | 99.64(0.11) | 92.73(1.28) |
| | TINc | 75.13(2.66) | 95.95(0.98) | 97.81(0.08) | 96.53(0.56) | 99.64(0.13) | 89.06(1.62) |
| | TINr | 72.70(2.85) | 95.26(1.22) | 98.92(0.07) | 96.87(0.62) | 99.60(0.10) | 87.62(1.79) |
| | SVHN | 64.30(4.74) | 91.52(2.61) | 90.93(0.48) | 94.01(0.98) | 97.97(0.20) | 91.60(0.93) |
| SVHN | CIFAR-10 | 95.39(0.44) | 97.10(1.20) | 91.23(0.85) | 99.66(0.03) | 99.87(0.02) | 100.00(0.00) |
| | CIFAR-100 | 95.34(0.57) | 96.48(1.27) | 92.04(0.80) | 99.67(0.02) | 99.77(0.02) | 100.00(0.00) |
| | Food-101 | 92.42(1.20) | 94.46(1.79) | 97.56(0.43) | 99.64(0.03) | 99.84(0.02) | 99.99(0.00) |
| | iSUN | 96.38(1.00) | 98.02(1.16) | 99.75(0.06) | 99.99(0.00) | 99.97(0.01) | 100.00(0.00) |
| | LSUNc | 95.21(0.93) | 97.89(0.86) | 97.41(0.07) | 99.72(0.02) | 99.87(0.04) | 99.96(0.00) |
| | LSUNr | 95.50(1.22) | 97.26(1.70) | 99.83(0.03) | 99.99(0.00) | 99.97(0.01) | 100.00(0.00) |
| | TINc | 96.75(0.95) | 98.11(1.10) | 99.21(0.12) | 99.96(0.01) | 99.97(0.01) | 100.00(0.00) |
| | TINr | 96.52(0.90) | 97.85(1.08) | 99.63(0.07) | 99.98(0.01) | 99.97(0.01) | 100.00(0.00) |
| Food-101 | CIFAR-10 | 84.66(0.13) | 97.36(0.23) | 84.80(0.37) | 92.28(0.38) | 95.03(1.77) | 100.00(0.00) |
| | CIFAR-100 | 83.56(0.20) | 96.97(0.26) | 88.02(0.44) | 93.76(0.31) | 95.29(1.62) | 100.00(0.00) |
| | iSUN | 85.07(1.36) | 98.65(0.56) | 99.47(0.10) | 99.84(0.05) | 99.77(0.07) | 96.50(1.09) |
| | LSUNc | 84.01(1.81) | 98.98(0.09) | 96.68(0.23) | 98.93(0.07) | 95.52(0.50) | 92.33(0.50) |
| | LSUNr | 83.73(1.28) | 98.32(0.70) | 99.45(0.10) | 99.81(0.06) | 99.81(0.06) | 95.85(1.40) |
| | TINc | 83.39(0.83) | 98.49(0.38) | 98.64(0.16) | 99.45(0.07) | 98.19(0.36) | 91.96(1.37) |
| | TINr | 82.98(0.92) | 98.18(0.57) | 99.10(0.12) | 99.61(0.09) | 99.21(0.20) | 93.58(1.60) |
| | SVHN | 65.59(2.17) | 96.06(1.28) | 99.19(0.17) | 99.69(0.09) | 99.35(0.23) | 99.80(0.11) |

# Chapter 5

# Conclusion

In this dissertation, we discuss the OOD detection in the classification problem. We first point out the importance of the OOD detection method independent of the hyperparameter tuning in Chapter 2. In the experiments, we show that the methods that rely on the hyperparameter tuning have compromised detection performance when the validation OOD is drawn from the different data distribution. This reflects the real-world assumption where it is difficult to guarantee the effectiveness of the OOD validation dataset. Our proposed method, i.e., cosine network, utilizes the scaled cosine softmax for training and uses cosine similarity for detecting OOD samples; it does not require the hyperparameter tuning. The proposed method outperforms the others in the less-biased evaluation [1] and shows competitive performance in the conventional evaluation, i.e., the validation dataset is allowed for the compared method. Although it performs well in OOD detection, its ID classification performance deteriorates. This problem can be alleviated using two networks, i.e., the standard network for classification and the cosine network for OOD detection.

We further question the existing OOD detection methods toward their effectiveness on the realistic image in Chapter 3. We divide the experiments into three scenarios, i.e., irrelevant inputs detection, novel class detection, and domain shift detection. *Irrelevant input detection* is similar to the OOD detection evaluation that is the most widely studied in the literature. It considers the case when the sample belongs to the class not included in the training dataset. *Novel class detection* resembles irrelevant input detection in that the sample belongs to the unknown class. However, it considers a more difficult case where the OOD samples are visually similar to the training dataset. *Domain shift detection* covers a different case when the model encounters the sample that belongs to the known classes; however, it is OOD due to the difference in the sample domain. In this scenario, the OOD score is expected to convey the expected deterioration of the model. We adopt the synthesized corrupted images to represent the domain shift detection. We train a small model to learn to predict the classification error w.r.t. the OOD score. As a result, we observe that the cosine network shows consistently good performance in all cases. Unexpectedly, we observe that the synthesized image has a similar deterioration trend to the realistic domain shifting, i.e., Office-31. This indicates that we can simulate the deterioration from the synthesized images.

Inspired by the observation in Chapter 3 about the synthesized corrupted images, we utilize them to train the model to improve the OOD detection result in Chapter 4. In Hendrycks et al. [7], the auxiliary dataset, i.e., outlier, is utilized to represent the seen OOD samples in training. This knowledge of the OOD is expected to be beneficial for detecting OOD in the inference time. The detection performance arguably depends on how to construct the outlier dataset. The decreasing of the classification performance is also observed in their method. We proposed to use the corrupted images to represent the OOD samples. As shown in Chapter 3, the corrupted images can be considered shifted dataset from the clean images. We apply each corruption to the original training dataset and record the classification accuracy. The recorded accuracy is then used to construct the soft-label target for training in the second attempt. We re-train the model and allow it to see the clean images and the corrupted images with soft-label mixed in the training batch. The re-trained model shows high performance in both classification and OOD detection, all-in-one. We also provide the analyses to show the effectiveness when the factors are varied and prove that having the validation OOD dataset is not necessary.

There are remaining problems in the study of OOD detection. Although many OOD detection methods have been proposed in the research community, the concrete understanding of the cause of the OOD sample is limited. It possibly gives more understanding to the other related problem as well, e.g., calibration and domain shifting. This understanding would be a crucial contribution to the community. The domain adaptation using more effective measurement to estimate the shift might contribute to performance improvement. These research fields would encourage the DNN to have more variety of applications with trust and safety in utilizing AI agents in the real world.

# Bibliography

[1] Alireza Shafaei, Mark Schmidt, and James J. Little. A less biased evaluation of out-of-distribution sample detectors. In *Proceedings of the British Machine Vision Conference*, 2019.

[2] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *Proceedings of the International Conference on Learning Representations*, 2017.

[3] Shiyu Liang, Yixuan Li, and R Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. In *Proceedings of the International Conference on Learning Representations*, 2017.

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

[5] Yen-Chang Hsu, Yilin Shen, Hongxia Jin, and Zsolt Kira. Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2020.

[6] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *Proceedings of the International Conference on Learning Representations*, 2019.

[7] Dan Hendrycks, Mantas Mazeika, and Thomas G Dietterich. Deep anomaly detection with outlier exposure. In *Proceedings of the International Conference on Learning Representations*, 2019.

[8] Chiara Longoni, Andrea Bonezzi, and Carey K Morewedge. Resistance to medical artificial intelligence. *Journal of Consumer Research*, 46(4):629–650, 2019.

[9] Thai-Son Nguyen, Sebastian Stueker, and Alex Waibel. Super-human performance in online low-latency recognition of conversational speech. *arXiv preprint arXiv:2010.03449*, 2020.

[10] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv:1606.06565*, 2016.

[11] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *Proceedings of the International Conference on Machine Learning*, 2017.

[12] Zhizhong Li and Derek Hoiem. Improving confidence estimates for unfamiliar examples. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2020.

[13] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the International Conference on Machine Learning*, 2016.

[14] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in Neural Information Processing Systems*, 2017.

[15] Christian Leibig, Vaneeda Allken, Murat Seçkin Ayhan, Philipp Berens, and Siegfried Wahl. Leveraging uncertainty information from deep neural networks for disease detection. *Scientific reports*, 7(1):17816, 2017.

[16] Terrance DeVries and Graham W Taylor. Leveraging uncertainty estimates for predicting segmentation quality. *arXiv:1807.00502*, 2018.

[17] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, 2017.

[18] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems*, 2019.

[19] Andrey Malinin and Mark Gales. Predictive uncertainty estimation via prior networks. In *Advances in Neural Information Processing Systems*, 2018.

[20] Terrance DeVries and Graham W Taylor. Learning confidence for out-of-distribution detection in neural networks. *arXiv:1802.04865*, 2018.

[21] Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. Training confidence-calibrated classifiers for detecting out-of-distribution samples. In *Proceedings of the International Conference on Learning Representations*, 2018.

[22] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? In *Proceedings of the International Conference on Learning Representations*, 2019.

[23] Hyunsun Choi and Eric Jang. Waic, but why? generative ensembles for robust anomaly detection. *arXiv:1810.01392*, 2018.

[24] Sungik Choi and Sae-Young Chung. Novelty detection via blurring. In *Proceedings of the International Conference on Learning Representations*, 2020.

[25] Chandramouli S Sastry and Sageev Oore. Detecting out-of-distribution examples with gram matrices. In *Proceedings of the International Conference on Machine Learning*, 2020.

[26] Sehun Yu, Dongha Lee, and Hwanjo Yu. Convolutional neural networks with compression complexity pooling for out-of-distribution image detection. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2020.

[27] Qing Yu and Kiyoharu Aizawa. Unsupervised out-of-distribution detection by maximum classifier discrepancy. In *Proceedings of the International Conference on Computer Vision*, 2019.

[28] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: A tutorial introduction. *arXiv:1910.09457*, 2019.

[29] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. In *International Conference on Learning Representations*, 2017.

[30] Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. Mvtec ad–a comprehensive real-world dataset for unsupervised anomaly detection. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2019.

[31] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[32] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, 2012.

[34] Gert Lanckriet, Laurent Ghaoui, Chiranjib Bhattacharyya, and Michael Jordan. Minimax probability machine. *Advances in neural information processing systems*, 14:801–807, 2001.

[35] Qilong Wang, Jiangtao Xie, Wangmeng Zuo, Lei Zhang, and Peihua Li. Deep cnns meet global covariance pooling: Better representation and generalization. *Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[36] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning*, 2015.

[37] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *Advances in Neural Information Processing Systems*, 2014.

[38] Apoorv Vyas, Nataraj Jammalamadaka, Xia Zhu, Dipankar Das, Bharat Kaul, and Theodore L Willke. Out-of-distribution detection using an ensemble of self supervised leave-out classifiers. In *Proceedings of the European Conference on Computer Vision*, 2018.

[39] Gabi Shalev, Yossi Adi, and Joseph Keshet. Out-of-distribution detection using multiple semantic label representations. In *Advances in Neural Information Processing Systems*, 2018.

[40] David Macêdo and Teresa Ludermir. Neural networks out-of-distribution detection: Hyperparameter-free isotropic maximization loss, the principle of maximum entropy, cold training, and branched inferences. *arXiv:2006.04005*, 2020.

[41] Rajeev Ranjan, Carlos D Castillo, and Rama Chellappa. L2-constrained softmax loss for discriminative face verification. *arXiv:1703.09507*, 2017.

[42] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2017.

[43] Feng Wang, Xiang Xiang, Jian Cheng, and Alan Loddon Yuille. Normface: L2 hypersphere embedding for face verification. In *Proceedings of the ACM International Conference on Multimedia*, 2017.

[44] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2018.

[45] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2019.

[46] Xiao Zhang, Rui Zhao, Yu Qiao, Xiaogang Wang, and Hongsheng Li. Adacos: Adaptively scaling cosine logits for effectively learning deep face representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[47] Hossein Azizpour, Mattias Teye, and Kevin Smith. Bayesian uncertainty estimation for batch normalized deep networks. In *Proceedings of the International Conference on Machine Learning*, 2018.

[48] Jochen Gast and Stefan Roth. Lightweight probabilistic deep networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2018.

[49] Hermann Blum, Paul-Edouard Sarlin, Juan Nieto, Roland Siegwart, and Cesar Cadena. Fishyscapes: A benchmark for safe semantic segmentation in autonomous driving. In *Proceedings of the International Conference on Computer Vision Workshops*, 2019.

[50] Petra Bevandić, Ivan Krešo, Marin Oršić, and Siniša Šegvić. Simultaneous semantic segmentation and outlier detection in presence of domain shift. In *Proceedings of the German Conference on Pattern Recognition*. Springer, 2019.

[51] Volodymyr Kuleshov, Nathan Fenner, and Stefano Ermon. Accurate uncertainties for deep learning using calibrated regression. In *Proceedings of the International Conference on Machine Learning*, 2018.

[52] Akshayvarun Subramanya, Suraj Srinivas, and R Venkatesh Babu. Confidence estimation in deep neural networks via density modelling. In *Proceedings of International Conference on Multimedia and Expo*, 2017.

[53] Walter J Scheirer, Anderson Rocha, Ross J Micheals, and Terrance E Boult. Meta-recognition: The theory and practice of recognition score analysis. *Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1689–1695, 2011.

[54] Tongfei Chen, Jiří Navrátil, Vijay Iyengar, and Karthikeyan Shanmugam. Confidence scoring using whitebox meta-models with linear classifier probes. In *Pro-*

*ceedings of the International Conference on Artificial Intelligence and Statistics*, 2018.

[55] Jie Ren, Peter J. Liu, Emily Fertig, Jasper Snoek, Ryan Poplin, Mark A. DePristo, Joshua V. Dillon, and Balaji Lakshminarayanan. Likelihood ratios for out-of-distribution detection. In *Advances in Neural Information Processing Systems*, 2019.

[56] Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. In *Proceedings of the International Conference on Machine Learning*, 2019.

[57] Pingmei Xu, Krista A Ehinger, Yinda Zhang, Adam Finkelstein, Sanjeev R Kulkarni, and Jianxiong Xiao. Turkergaze: Crowdsourcing saliency with webcam based eye tracking. *arXiv:1504.06755*, 2015.

[58] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Proceedings of the International Conference on Learning Representations*, 2015.

[59] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2014.

[60] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2015.

[61] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2006.

[62] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Sparsifying neural network connections for face recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2016.

[63] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. Large-margin softmax loss for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning*, 2016.

[64] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2009.

[65] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv:1506.03365*, 2015.

[66] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *Advances in Neural Information Processing Systems Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[67] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *Proceedings of the European Conference on Computer Vision*, 2014.

[68] Adam Coates, Honglak Lee, and Andrew Y. Ng. An analysis of single layer networks in unsupervised feature learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2011.

[69] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 1998.

[70] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv:1708.07747*, 2017.

[71] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference*, 2016.

[72] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2017.

[73] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Canadian Institute for Advanced Research, 2009.

[74] Engkarat Techapanurak, Suganuma Masanori, and Takayuki Okatani. Hyperparameter-free out-of-distribution detection using softmax of scaled cosine similarity. *arXiv:1905.10628*, 2019.

[75] Hady Elsahar and Matthias Gallé. To annotate or not? predicting performance drop under domain shift. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing*, 2019.

[76] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):1–37, 2014.

[77] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *Proceedings of the International Conference on Machine Learning*, 2015.

[78] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2017.

[79] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[80] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. In *Proceedings of the International Conference on Computer Vision*, 2019.

[81] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2016.

[82] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization: Stanford dogs. In *Proceedings of the Conference on Computer Vision and Pattern Recognition Workshop on Fine-Grained Visual Categorization*, 2011.

[83] Thomas Mosgaard Giselsson, Rasmus Nyholm Jørgensen, Peter Kryger Jensen, Mads Dyrmann, and Henrik Skov Midtiby. A public image database for benchmark of plant seedling classification algorithms. *arXiv:1711.05458*, 2017.

[84] Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florain Schroff, Serge Belongie, and Pietro Perona. Caltech-UCSD Birds 200. Technical report, California Institute of Technology, 2010.

[85] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the International Workshop on 3D Representation and Recognition*, 2013.

[86] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2012.

[87] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In *Advances in Neural Information Processing Systems*, 2007.

[88] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *Proceedings of the European Conference on Computer Vision*, 2010.

[89] Ev Zisselman and Aviv Tamar. Deep residual flow for out of distribution detection. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, June 2020.

[90] Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. Moment matching for multi-source domain adaptation. In *Proceedings of the International Conference on Computer Vision*, 2019.

[91] Yang Zhang, Philip David, and Boqing Gong. Curriculum domain adaptation for semantic segmentation of urban scenes. In *Proceedings of the International Conference on Computer Vision*, 2017.

[92] Marco Toldo, Andrea Maracani, Umberto Michieli, and Pietro Zanuttigh. Unsupervised domain adaptation in semantic segmentation: a review. *arXiv:2005.10876*, 2020.

[93] Yang Zou, Zhiding Yu, Xiaofeng Liu, BVK Kumar, and Jinsong Wang. Confidence regularized self-training. In *Proceedings of the International Conference on Computer Vision*, 2019.

[94] 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. *arXiv preprint arXiv:2009.11732*, 2020.

[95] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *Proceedings of the International Conference on Learning Representations*, 2019.

[96] Patrick McClure and Nikolaus Kriegeskorte. Robustly representing uncertainty in deep neural networks through sampling. In *Advances in Neural Information Processing Systems Workshop on Bayesian Deep Learning*, 2017.

[97] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2016.

[98] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2019.

[99] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. In *Proceedings of International Conference on Learning Representations Workshop*, 2017.

[100] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2018.

[101] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. When does label smoothing help? In *Advances in Neural Information Processing Systems*, pages 4694–4703, 2019.

[102] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In *Proceedings of the International Conference on Machine Learning*, 2018.

[103] Daiki Tanaka, Daiki Ikami, Toshihiko Yamasaki, and Kiyoharu Aizawa. Joint optimization framework for learning with noisy labels. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2018.

[104] Kun Yi and Jianxin Wu. Probabilistic end-to-end noise correction for learning with noisy labels. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2019.

[105] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

[106] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations*, 2019.

[107] Dan Hendrycks, Norman Mu, Ekin Dogus Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple method to improve robust-

ness and uncertainty under data shift. In *International Conference on Learning Representations*, 2020.

[108] Evgenia Rusak, Lukas Schott, Roland S Zimmermann, Julian Bitterwolf, Oliver Bringmann, Matthias Bethge, and Wieland Brendel. A simple way to make neural networks robust against diverse image corruptions. In *Proceedings of the European Conference on Computer Vision*, 2020.

[109] 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.

# List of Publications

## Proceedings of International Conferences

1. **Engkarat Techapanurak**, Masanori Suganuma, and Takayuki Okatani, “Hyperparameter-free out-of-distribution detection using cosine similarity”, *Proceedings of the Asian Conference on Computer Vision (ACCV)*, 2020

## Non-Peer Reviewed Publication

1. **Engkarat Techapanurak**, Takayuki Okatani, “Practical evaluation of out-of-distribution detection methods for image classification”, *arXiv preprint arXiv:2101.02447*, 2021. Under submission to the International Conference of Computer Vision (ICCV) 2021.

2. **Engkarat Techapanurak**, Anh Chuong Dang, Takayuki Okatani, “Bridging in- and out-of-distribution samples for their better discriminability”, *arXiv preprint arXiv:2101.02500*, 2021. Under submission to the Conference on Computer Vision and Pattern Recognition (CVPR) 2021.

3. Liang Xu, Taro Hatsutani, Xing Liu, **Engkarat Techapanurak**, Han Zou, Takayuki Okatani, “Pushing the envelope of thin Crack detection”, *arXiv preprint arXiv:2101.03326*, 2021.

# Acknowledgments

This dissertation would not have been finished without many people surrounding me. It is not only physically but also those who send their warmth from far away. It must be said that I survived in a difficult situation with their kindness and support. I will casually write this section aiming to express everything honestly.

I was out of the Deep Learning field when I started working on this topic. I was wondering at that time how can I be accepted into this laboratory. Fortunately, Professor Okatani Takayuki allowed me to learn and improve myself. As my supervisor, he needed to be patient with my stupidity many times. I still remember when I presented him with a result of my wrong coding. However, he has never given up pushing me in the right direction. Without his support and supervision, I cannot imagine how I possibly finish this research. I thank him so much.

I also thank Assistant Professor Suganuma Masanori and Research Assistant Professor Liu Xing. They are both friends and teachers who give many helpful suggestions about both research and daily life. There was a time Professor Suganuma helped delivering me my forgotten key from the laboratory very late at night. Without his help, I would have been in trouble on that day. Professor Liu always answers my questions, no matter how many they are.

I got a lot of help from Mrs. Sakane Akemi, the secretary of the laboratory. She facilitates every member of the laboratory with her kindness. All the activities in the laboratory have never been done without her support.

All the textbooks I studied at the beginning of my journey were suggested by former Research Assistant Professor Mete Ozay. Those books provide a fundamental knowledge that later becomes a fundamental of my research too.

I luckily have many friends during this period. Fortunately, I got help from Dang Anh Chuong, especially for the work in Chapter 4. When I get into trouble with mathematics theory, Liu Kang-Jun answers all questions, no matter what they are. Van Quang Nguyen shows me how a successful person looks like. Hu Junjie suggests to me how to handle the bad time. Xu Liang recommended me wonderful ramen. Luo Xiyang always tells me how to avoid COVID-19. Nimar Blume and Davis Sorenson always correct my poor English. I also got kind support from Murase Rito, who was my tutor in the first semester.

I thank all of the friends in the laboratory, regardless of their nationality. Also, I thank all the friends I have outside the campus. Although I do not mention everyone, please be reminded that I am so grateful to spend time with them. An excellent moment cannot be possible without them.

Not only the people who physically stay near me, but I also thank my friends and family from far away. They are forced to listen to me whenever I was in trouble.

Many lessons have been taught by failure. I also thank myself in the past who produce the mistakes. And I hope myself in the future will not produce the mistakes as many as in the past. That is the meaning of *learning*. For any errors or inadequacies that may remain in this work, of course, the responsibility is entirely my own.