A Robust Classification-autoencoder to Defend Outliers and Adversaries*

Lijia Yu and Xiao-Shan Gao
Academy of Mathematics and Systems Science, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Email: xgao@mmrc.iss.ac.cn

Abstract
In this paper, a robust classification-autoencoder (CAE) is proposed, which has strong ability to recognize outliers and defend adversaries. The main idea is to change the autoencoder from an unsupervised learning model into a classifier, where the encoder is used to compress samples with different labels into disjoint compression spaces and the decoder is used to recover samples from their compression spaces. The encoder is used both as a compressed feature learner and as a classifier, and the decoder is used to decide whether the classification given by the encoder is correct by comparing the input sample with the output. Since adversary samples are seemingly inevitable for the current DNN framework, the list classifier to defend adversaries is introduced based on CAE, which outputs several labels and the corresponding samples recovered by the CAE. Extensive experimental results are used to show that the CAE achieves state of the art to recognize outliers by finding almost all outliers; the list classifier gives near lossless classification in the sense that the output list contains the correct label for almost all adversaries and the size of the output list is reasonably small.

Keywords. Robust DNN, classification-autoencoder, list classifier, decouple classification, outlier, adversary sample.

1 Introduction
The deep neural network (DNN) [18] has become the most powerful machine learning method, which has been successfully applied in computer vision, natural language processing, autonomous driving, and many other fields. On the other hands, the DNN still has weaknesses for improvements, such as the lack of explainability and robustness [9].

Robustness is a key desired feature for DNNs. In general, a DNN is said to be robust, if it can not only correctly classify samples containing noises, but also has the ability to recognize outliers and to defend adversaries [11 3 8 45].

Outlier detection is a key issue in the open-world classification [45 32], where the inputs to the DNN are not necessarily satisfy the same distribution with the training data set. On the contrary, the objects to be classified usually consist of a low-dimensional subspace of the total input space to the DNN and the majorities of the inputs are outliers. To be more precise, let us consider a classification DNN $F : \mathbb{I}^n \rightarrow \mathbb{L}$ for certain object $\mathcal{O} \subset \mathbb{I}^n$, where $\mathbb{I} = \{0, 1\}$ and $\mathbb{L} = \{0, 1, \ldots, o\}$ is the label set. For the MNIST dataset,

---

*This work is partially supported by the NKRDP grants No.2018YFA0704705, No.2018YFA0306702 and the NSFC grant No.12288201.
is the hand-written numbers represented by images in $\mathbb{I}^{28 \times 28}$, $\mathbb{L} = \{0, 1, \ldots, 9\}$, and $n = 784$. In general, $\mathbb{O}$ is considered to be a very low-dimensional subset of $\mathbb{I}^{n}$ and hence the majorities of elements in $\mathbb{I}^{n}$ are not in $\mathbb{O}$, which are called outliers. However, for each element $x \in \mathbb{I}^{n}$, a trained $\mathcal{F}$ will give a label in $\mathbb{L}$ to $x$, which is wrong with high probability if $\mathcal{F}$ is not specifically designed and trained to defend outliers.

A more subtle and difficult problem related to the robustness of DNN is the existence of adversaries [6, 35], that is, it is possible to intentionally make little modification to an image in $\mathbb{O}$ such that human can still recognize the object clearly, but the DNN outputs a wrong label or even any label given by the adversary. Existence of adversary samples makes the DNN vulnerable in safety-critical applications. Although many effective methods for training DNN to defend adversaries were proposed [1, 3], it was shown that adversaries seem still inevitable for current DNNs [2, 5, 29].

There exist vast literatures on improving the robustness of DNNs [1, 3, 8, 45]. In this paper, we present a new approach by changing the autoencoder from an unsupervised learning model into a classifier.

1.1 Contribution

In this paper, we present a DNN which has strong ability to recognize outliers and defend adversaries. The basic idea is to change the autoencoder from an unsupervised learning model into a classification network. The encoder $\mathcal{E}$ of the autoencoder is used to compress the input images into a low-dimensional space $\mathbb{R}^{m}$ ($m \ll n$) such that images with the same label are compressed into the compression space of that label and images with different labels are compressed into disjoint subsets of $\mathbb{R}^{m}$, called compression spaces. Furthermore, the images in $\mathbb{O}$ can be approximately recovered by the decoder $\mathcal{D}$ from their compression spaces. The encoder $\mathcal{E}$ is used both as a feature learner and a coarse classifier, which is different from the usual classifiers in that several coordinates instead of one are used to classify as well as to represent the images for each label. The decoder $\mathcal{D}$ can be used to give the final classification by comparing the input image with the output.

The above network is called a classification-autoencoder (CAE). We prove that such a network exists in certain sense. Precisely, we prove that there exists an autoencoder such that the encoder compresses images with different labels into disjoint compression sets of $\mathbb{R}^{m}$ for any $m$ and the decoder approximately recovers the input image from its compression set with any given precision.

The CAE is evaluated in great detail using numerical experiments. It is shown that the CAE achieves state of the art to recognize outliers by finding almost all outliers robustly. As an autoencoder, the outputs of a CAE are always like the object $\mathbb{O}$ to be classified. By definition, an outlier is an image which is not considered to be an element of $\mathbb{O}$, so an image is treated as an outlier if the input and the output are different. The CAE also works well for adversaries in the following sense. For a large proportion of adversaries of the encoder-classifier, the CAE can recognize them as problem images, that is, they are outliers or adversaries. Since adversaries are seemingly inevitable for DNNs [2, 29], a possible way to alleviate the problem is to give several answers instead of one. In a CAE, we can apply the decoder $\mathcal{D}$ to the compression space of each label to recover images and output the labels and the corresponding images which are similar to the input image. This kind of classifier is called list classifier and the output is a classification list, which tries to give uncertain but lossless classifications.

Our experimental results show that for almost all adversaries, the classification list contains the correct label and the size of the output list is reasonably small. A lossless classification does not miss important information, which is important for safety-critical applications. Furthermore, the classification list can be used for further analysis. For instance, the LCAE can be used to do decouple classification, which means to recover one or more elements from a sample containing more than two well-mixed elements of $\mathbb{O}$.
1.2 Related work

The autoencoder is one of the most important neural networks for unsupervised learning [4], which has many improvements and applications. The autoencoder learns compressed features in a low-dimensional space for high-dimensional data with minimum reconstruction loss, while our CAE makes classification at the same time of learning features. It is natural to use autoencoders for outlier detection due to its reconstruction property [46]. We improve this in two aspects. First, by compressing images with different labels into disjoint compression spaces, the robustness is increased and the classification can be given. Second, we use the compression spaces to introduce the list classification to increase the robustness. The ladder network, which is a variant of autoencoder, was used to classify the input images [28]. Our work is different in two aspects. First, a different DNN structure is introduced in this paper and thus the loss functions are different. Second, the work in [28] was mainly focused on denoising and our work is mainly for defending adversaries and outliers. The robustness of autoencoder was studied [17, 46, 26]. In principle, these methods can be applied to our CAE model to further improve the robustness.

A simple approach to recognize outliers is to introduce a new label representing outliers and add outlier samples [45]. The difficulty with this approach is that the distribution of outliers is usually too complex to model in high dimensional spaces. As a consequence, a network based on this approach works well for those outliers similar to that in the training set and works poorly for other outliers, as shown by the experimental results in this paper. Many approaches were proposed to detect outliers [32, 8, 45]. On the other hand, the CAE proposed in this paper is more natural to detect outliers, because the output of the CAE are assumed to be similar to the objects to be classified.

Many methods were proposed to train more robust DNNs to defend adversaries [40]. In the adversary training method proposed by Madry et al, the value of the loss function at the worst adversary in a small neighborhood of the training sample is minimized [22]. This approach can reduce the adversaries significantly [22, 44]. A similar approach is to generate adversaries and add them to the training set [13]. Another major approach is the gradient obfuscation methods, which deliberately hide or randomize the gradient information of the model, so that gradient based attacks cannot be used directly [7, 14, 31, 33, 38, 43]. The ensembler adversarial training [36] was introduced for CNN models, which can apply to large datasets such as ImageNet. A fast adversarial training algorithm was proposed, which improves the training efficiency by reusing the backward pass calculations [30]. A less direct approach to enhance the ability for the network to resist adversaries is to make the DNN more stable by introducing the Lipschitz constant or \( L_{2,\infty} \) regulations of each layer [10, 35, 42].

Effective methods were proposed to train more robust DNNs to defend adversaries [40]. In the adversary training method proposed by Madry et al, the loss function is a min-max optimization problem such that the value of the loss function of the worst adversary in a small neighborhood of the training sample is minimized [22]. This approach can reduce the adversaries significantly [22, 44]. Another similar approach is to generate adversaries and add them to the training set [13]. The ensembler adversarial training [36] was introduced for CNN models, which can apply to large datasets such as ImageNet. A fast adversarial training algorithm was proposed, which improves the training efficiency by reusing the backward pass calculations [30]. A less direct approach to enhance the ability for the network to resist adversaries is to make the DNN more stable by introducing the Lipschitz constant or \( L_{2,\infty} \) regulations of each layer [10, 35, 42].

Increase the robustness of the network in general will increase its ability to defend adversaries and there exist quite a lot of work on robust DNNs [16, 45]. Adding noises to the training data is an effective way to increase the robustness [12, Sec.7.5]. The \( L_1 \) regulation and \( L_{1,\infty} \) normalization are used to increase the robustness of DNNs [41]. Knowledge distilling is also used to enhance robustness and defend adversarial examples [16]. In [15, 21, 27, 39], methods to compute the robust regions of a DNN were given.
The rest of this paper is divided into four parts. In section 2 we give the structure for the CAE and prove its existence. In section 3 we give experimental results and show that the CAE can recognize almost all outliers. In section 4 we give the list classification algorithm and the experimental results for it to defend adversaries. In section 5 conclusions are given.

## 2 Structure and existence of the classification-autoencoder

### 2.1 The main idea

Let $I = [0, 1]$ and $F$ an autoencoder

$$F = D \circ E : \mathbb{I}^n \rightarrow \mathbb{I}^n \quad (1)$$

with encoder $E : \mathbb{I}^n \rightarrow \mathbb{R}^m$ and decoder $D : \mathbb{R}^m \rightarrow \mathbb{I}^n (m \ll n)$. $F$ is called a classification-autoencoder (CAE) for a classification problem, if $E$ compresses an input sample with different labels into disjoint compression spaces in $\mathbb{R}^m$ and $D$ recovers the input images to any given precision from their compression spaces.

As shown in Figure 1 for an input image $x$, let $y_l$ be the projection of $E(x)$ to the compression space of label $l$. Then the encoder gives a potential label $l$ to $x$ if $y_l$ has larger weight than other $y_i$ for $i \neq l$. Furthermore, if $F(x)$ is very similar to $x$, then we believe that $x$ is a normal sample with label $l$. On the contrary, if $F(x)$ not similar to $x$, then $x$ is either an outlier or an adversary. In this way, the CAE can be used as an open-world classifier by doing classification for normal samples as well as recognizing outliers and adversaries.

### 2.2 Structure and training of the classification-autoencoder

#### 2.2.1 The classification-encoder $E$

The encoder $E$ will be used both as a compressor as well as a classifier. Let $L = \{0, 1, \ldots, o\} \subset \mathbb{N}$ be the set of labels and $m = (o + 1)m_0$, where $m_0 \in \mathbb{N}_+$ is a hyperparameter to be defined by the user. We try to use $E$ to compress an image with label $l$ to the compression space of label $l$, which consists of the $k$-th coordinates of $\mathbb{R}^m$ for $k = lm_0 + 1, \ldots, lm_0 + m_0$. For $l \in L$, define the compression mask vectors $M_l \in \mathbb{R}^m$ for the $l$-th compression space as follows

$$M_l[j] = \begin{cases} 1, & \text{when } lm_0 + 1 \leq j \leq lm_0 + m_0 \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$
For $y \in \mathbb{R}^m$ and $l \in L$, the projection weight of $y$ to the $l$-th compression space is

$$W_l(y) = y \cdot M_l$$

where $\cdot$ is the inner product.

Let $x \in \mathbb{I}^n$ be a sample in the training set and $l_x \in L$ its label. In order to send $x$ to its compression space, we define the following loss function for $E$

$$\text{Loss}_{E}(x, l_x) = \text{Loss}_{CE}(A(x)/\gamma, l_x)$$

where $A(x) = \{W_0(E(x)), \ldots, W_o(E(x))\}$ and $\gamma$ is a hyperparameter. Intuitively, the loss function means that the weight of $E(x)$ in the compression space for label $l_x$ will be maximized.

### 2.2.2 The classification-autoencoder

The decoder $D$ in (1) tries to recover $x$ from its compression subspace for $x \in \mathbb{I}^n$. We first define a projection function

$$p(y, l) = y \circ M_l : \mathbb{R}^m \times L \to \mathbb{R}^m$$

where $\circ$ is the Hadamard (element-wise) product and $M_l$ is the mask vector defined in (2). Note that $p(y, l)$ projects $y$ to the compression space of label $l$.

Then the total network $F$ is certain composition of $E$ and $D$:

$$F(x) = D(p(E(x), l_x)) = D(E(x) \circ M_{l_x})$$

and the loss function for $F$ is

$$\text{Loss}(x, l_x) = \text{Loss}_{E}(x, l_x) + \lambda \text{Loss}_{MSE}(F(x), x)$$

where $\lambda$ is a hyperparameter.

Moreover, to increase the robustness of the CAE, we can use adversarial training [22] and the loss function for $F$ becomes

$$\text{Loss}(x, l_x) = \text{Loss}_{E}(x + \eta_x, l_x) + \lambda(\text{Loss}_{MSE}(F(x + \eta_x), x) + \text{Loss}_{MSE}(F(x), x))$$

where $\eta_x = \arg \max ||\eta||_\infty < \epsilon \text{Loss}_{E}(x + \eta, l_x)$.

**Remark 2.1.** The loss function is a joint loss over the classification space as well as a reconstruction loss on the decoder phase, which makes $F$ both as a classifier and as a feature learner. The CAE is different from the usual classifiers in that several coordinates instead of one are used to classify as well as to learn features.

Given a set of training samples, we can train $F$ with the standard gradient descendent method based on the above loss functions. In what below, we give a termination criterion for the training algorithm. Since a sample $x$ with label $l_x$ should satisfy $C(x) = 1$, we have $E(x) \cdot M_{l_x} - E(x) \cdot M_i \geq C_0$ for $i \neq l_x$. Thus

$$\text{Loss}_{E}(x, l_x) = -\ln \frac{e^{E(x)\cdot M_{l_x}/\gamma}}{\sum_{i=0}^o e^{E(x)\cdot M_i/\gamma}} = -\ln \frac{1}{\sum_{i=0}^o e^{-(E(x)\cdot M_{l_x} - E(x)\cdot M_i)/\gamma}} \leq \ln (1 + 9e^{-C_0/\gamma}).$$

So when $\text{Loss}_{E}(x, l_x) \leq \ln (1 + 9e^{-C_0/\gamma})$ for most $(x, l_x)$ in the training set and $\text{Loss}(x, l_x)$ is small enough, we terminate the algorithm.
### 2.3 The open-world classification algorithm

Let \( \mathcal{F} \) be a trained CAE with the loss functions (6) or (7). In this section, we show how to use \( \mathcal{F} \) as an open-world classifier. Precisely, the algorithm will return a label in \( \mathcal{L} \) if the input is considered to be a normal sample and return label \(-1\) if the input is considered to be a problem image. A problem image could be either an outlier or an adversary.

For an input \( x \in \mathbb{I}^n \), according to the loss function in (4), the pseudo-label of \( x \) is

\[
L(x) = \arg \max_{l \in \mathcal{L}} W_l(\mathcal{E}(x)) : \mathbb{R}^n \to \mathcal{L}.
\]

(8)

With the pseudo-label \( L(x) \), the output of the network is defined as

\[
\hat{\mathcal{F}}(x) = \mathcal{D}(p(\mathcal{E}(x), L(x))) = \mathcal{D}(\mathcal{E}(x) \circ M_{L(x)}).
\]

(9)

The main idea of the algorithm is to compare \( x \) and \( \hat{\mathcal{F}}(x) \) to see if \( x \) is a normal sample or a problem image.

**Algorithm 1 CAE**

| Input: \( x \in \mathbb{I}^n \), hyperparameters \( C_0, b_s, b_u \) in \( \mathbb{R}^+ \). |
| Output: a label of \( x \) in \( \mathcal{L} \) or label \(-1\) meaning that \( x \) is a problem image. |

**S1** Compute \( \mathcal{F}(x) \) with (5), \( L(x) \) with (8), and \( \hat{\mathcal{F}}(x) \) with (9).

**S2** If \( \|\hat{\mathcal{F}}(x) - x\| \leq b_s \) for a given threshold \( b_s \in \mathbb{R}^+ \), then output: label \( L(x) \).

**S3** If \( \|\mathcal{F}(x) - x\| \geq b_u \) for a given threshold \( b_u \in \mathbb{R}^+(b_s < b_u) \), then output label \(-1\).

**S4** This step treats the \( x \) satisfying \( b_s < \|\hat{\mathcal{F}}(x) - x\| < b_u \).

**S4.1** Let \( d_l = \|\mathcal{D}(p(\mathcal{E}(x), l)) - x\| \), for all \( l \in \mathcal{L} \).

**S4.2** If \( d_{L(x)} \leq d_l \) for \( l \in \mathcal{L} \), then output label \( L(x) \).

**S4.3** Output label \(-1\).

We explain the algorithm as follows. Let \( \mathcal{O} \) be the set of images to be classified. In Step S2, when \( \hat{\mathcal{F}}(x) \) and \( x \) are similar enough, we think \( x \) is in \( \mathcal{O} \) and its label is given. In Step S3, when \( x \) is not anything like \( \hat{\mathcal{F}}(x) \), \( x \) is a problem image. In Step S4, we are not sure whether \( x \) is an element in \( \mathcal{O} \) or a problem image. In this case, we give a refined analysis by computing \( \mathcal{D}(p(\mathcal{E}(x)), l) \) for all \( l \in \mathcal{L} \) and checking if \( \mathcal{D}(p(\mathcal{E}(x)), L(x)) \) is more similar to \( x \) than \( \mathcal{D}(p(\mathcal{E}(x)), l) \).

**Remark 2.2.** *The input to Algorithm [7] could be a sample in \( \mathcal{O} \) or a problem image. By the accuracy of the algorithm on an input set, we use the usual meaning, that is, the percentage of samples which are given the correct labels in \( \mathcal{L} \). By the total accuracy of the algorithm on an input set, we mean the percentage of inputs which are in \( \mathcal{O} \) and are given the correct labels and the inputs which are problem images and are considered as problem images.*

### 2.4 Existence of the CAE

The main idea of CAE is that images with the same label are mapped into a compression space and images with different labels are compressed into disjoint compression spaces by the encoder. Furthermore, the
images to be classified can be approximately recovered from their compression spaces. In this section, we prove that DNNs with such properties exist in certain sense.

Let \( x \in \mathbb{R}^n \) and \( r \in \mathbb{R}^+ \). When \( r \) is small enough, all images in

\[
B(x, r) = \{ x + \eta \mid \eta \in \mathbb{R}^n, ||\eta|| < r \}
\]

can be considered to have the same label with \( x \). Therefore, the set of images \( \emptyset \) to be classified can be considered as bounded open sets in \( \mathbb{R}^n \). This observation motivates the existence result given below, where \( V(S) \) is used to denote the volume of \( S \subset \mathbb{R}^n \).

**Theorem 1.** Suppose that the objects to be classified consist of a finite number of disjoint and bounded open sets \( S_l \subset \mathbb{I}^n \) such that objects in \( S_l \) have label \( l \in \mathbb{L} = \{0, \ldots, o\} \). Then, for any \( m \in \mathbb{N}_+ \) and for any \( \epsilon, \gamma \in \mathbb{R}^+ \), there exist DNNs \( E : \mathbb{I}^n \rightarrow \mathbb{R}^m \) and \( D : \mathbb{R}^m \rightarrow \mathbb{I}^n \) such that

1. Let \( K = \{ x \in \bigcup_{l=0}^{o} S_l \mid E(x) \in E(S_l) \cap E(S_k) \text{ for some } l \neq k \} \). Then \( \frac{V(K)}{\sum_l V(S_l)} < \epsilon \).

2. Let \( \tilde{S}_l = \{ x \in S_l \mid ||D(E(x)) - x|| < \gamma \} \) for \( l \in \mathbb{L} \). Then \( \frac{V(\tilde{S}_l)}{V(S_l)} > 1 - \epsilon \).

*Proof.* We give the idea of the proof and the detail of the proof is given in Appendix A. Let \( \mathbb{I}_0 = (0, 1) \), \( A_0 = \mathbb{I}_0^m \), and \( A_j = \{ y \mid y = z + 2j1, z \in A_{j-1} \} \) for \( j = 1, \ldots, o \), where \( 1 \in \mathbb{R}^m \) is the vector whose coordinates are 1. Then we can define piece-wise constant functions \( E : \mathbb{I}^n \rightarrow \mathbb{R}^m \) and \( D : \mathbb{R}^m \rightarrow \mathbb{I}^n \), such that \( E \) is approximately the identity map from \( S_l \) to \( A_l \) with \( D \) the inverse map for \( l \in \mathbb{L} \), that is \( A_l \) is the compression space of label \( l \). Finally, based on the universal approximation property of DNNs, the required autoencoder can be constructed. \( \square \)

**Remark 2.3.** Theorem 1 implies that different \( S_l \) are mapped into approximately disjoint spaces \( E(S_l) \cong A_l \) and almost all elements in \( S_l \) can be approximately recovered from \( E(S_l) \) by \( D \), and thus proves the existence of the CAE.

**Remark 2.4.** In Theorem 1, \( m \) could be as small as possible, and \( E(S_l) \) are contained in disjoint unit cubes in \( \mathbb{R}^m \). In practice, in order that the network can be trained efficiently, we assume \( E(S_l) \subset \mathbb{R}^{m_l} \) for some \( m_l \in \mathbb{N}_+ \), \( m = \sum_{l=0}^{o} m_l \), and \( \mathbb{R}^m \) the direct product of these \( \mathbb{R}^{m_l} \).

## 3 Experimental results for CAE

In this section, we will give experimental results based on MNIST dataset in great detail. The precise structures of the networks used in the experiments are given in the Appendix B. The hyperparameters are \( \gamma = 50, \lambda = 1, C_0 = 80, b_l = 0.04 \), and \( b_u = 0.09 \). The codes can be found at https://github.com/yyyylllj/NetB.

### 3.1 Accuracy on the test set

We first show that the CAE works well for the test set as a classifier. Among the 10000 samples in the test set of MNIST, 9864 samples are given the correct labels, 43 samples are given the wrong labels, and 93 samples are considered as problem images. Some of the samples which are give the wrong label or considered as problem images are given in Figure 2.
Figure 2: The images in left-side part are the input images and the ones in the right-side part are the corresponding output of CAE. Row 1: problem images. Row 2: images for which wrong labels are given.

| Data Set | CAE     | LCAE    | NLabels |
|----------|---------|---------|---------|
| MNIST    | 98.64%  | 99.97%  | 1.72    |

Table 1: Accuracies for CAE and and LCAE (section 4.2). NLabels is average number of labels in the output list of LCAE.

Remark 3.1. As mentioned above, 39 samples are given the wrong labels and 131 samples are considered as problem images. As we can see in Figure 2, the quality of these images is quite poor and they indeed can be considered as problem images. This property is not all bad and can even be used to identify bad samples in a data set. A solution to recognize almost all of these problem images is given in section 4 and the result is given in the column LCAE in Table 1.

3.2 Recognize outliers

In this section, we use four types of outliers to check the ability of \( F \) to recognize outliers.

The type 1 outliers are images in \( \mathbb{R}^{28 \times 28} \), whose coordinates are generated by \( D_n \) (iid). Here, \( D_n = \frac{1}{2}(\mathcal{N} + \mathcal{U}) \), where \( \mathcal{N} \) is the normal distribution and \( \mathcal{U} \) is the uniform distribution.

The type 2 outliers are random samples with structures. They are created by randomly generating a row, a column, or the diagonal, and then by copying the row, column or the diagonal randomly to cover the matrix.

For the type 3 outliers, the middle (12 \( \times \) 12) 144 pixels of an image in \( \mathbb{R}^{28 \times 28} \) are given by the normal distribution \( \mathcal{N} \) (iid).

The type 4 outliers for MNIST are generated by reducing the sizes of images in CIFAR-10 to 28 \( \times \) 28 and change the image from color to grey. For CIFAR-10, type 4 outliers are obtained from MNIST similarly.

Figure 3: The four types of outliers (left-side) for MNIST and theirs outputs (right-side) from \( F \) are given in the four rows from type 1 to type 4.
We compare our network with a CNN $H$ which is trained with MNIST plus 60000 noise samples with a new label $-1$, representing the class of outliers. The noise samples are generated by $\alpha N + \beta U$, where $\alpha$ is a random number in $(0, 1)$ and $\beta = 1 - \alpha$.

We compare $F$ and $H$ and the results are given in the first four rows of Table 2. As shown in Figure 3, for an outlier $x$, $x$ and $F(x)$ are quite different, which means $||F(x) - x||$ is big enough to make the network treating $x$ as a problem image. As a consequence, our network can recognize all kind of outliers. On the other hand, $H$ can only recognize the outliers similar to the training outlier samples.

We also compare the robustness of $F$ and $H$ to recognize outliers. Let $x_o$ be a type 1 outlier. Two types of new outliers are generated from $x_o$ as follows. For $H$, we use gradient descent for $x_o$ to make $L_{CE}(x_o, -1)$ smaller. For $F$, we use gradient descent for $x_o$ to make $L_{MSE}(D(p(E(x_o), L(x_o))) - x_o)$ smaller. Use Type 1.i to denote the new outliers, where each pixel of $x_o$ is changed up to 0.i for $i = 1, 2$ and the results are listed in the fifth and sixth rows of Table 2. From Table 2 we can see that $F$ is very robust to recognize this kind of strong outliers, while $H$ is much less robust.

| Outliers  | CAE(M) | Network $H$(M) | CAE(C) | Network $H$(C) |
|-----------|--------|----------------|--------|----------------|
| Type 1    | 100%   | 99%            | 100%   | 99%            |
| Type 2    | 100%   | 97%            | 100%   | 92%            |
| Type 3    | 100%   | 3%             | 100%   | 23%            |
| Type 4    | 100%   | 26%            | 100%   | 40%            |
| Type 1.1  | 100%   | 80%            | 100%   | 10%            |
| Type 1.2  | 100%   | 20%            | 100%   | 5%             |

Table 2: Percentages for CAE and $H$ to recognize outliers or the total accuracy defined in Remark 2.2. (C) means experiment in CIFAR-10, (M) means MNIST.

**Remark 3.2.** Types 1, 2, 3 are all outliers. We use them to show that a network trained with samples outliers works well for samples similar to the training set, but works poorly for samples different from the training set. For instance, $H$ works well for type 1 and type 2 outliers, but it works poor for type 3 outliers. The type 5 adversarial outliers are also used to show that there exist no exact boundaries between outliers and normal samples satisfying the distribution and outliers.

To summarize, the CAE can recognize almost all outliers robustly and this is one of the main advantages of the network introduced in this paper.

### 3.3 Defend adversaries

In this section, we check the ability of $F$ to defend adversaries. For $F$, $E(x)$ is the network for classification, so we create adversaries for $E(x)$. We use samples in the training set to generate 3 types of adversaries.
The type 1.i ($i = 10, 20, 30$) adversaries ($L_{\text{inf}}$ adversary) are created with PGD-$i$ \cite{22}, where each step changes 0.01 for every pixel and at most $i$ steps are used.

The type 2.i ($i = 40, 60, 80$) adversaries ($L_0$ adversary) are obtained with JSMA \cite{24} by changing $i$ coordinates of a training sample, and each coordinate can change at most 1.

The type 3.i ($i = 0, 1, 2$) adversaries are called strong adversaries which are generated by two steps. First, generate a type 1.20 adversary $x_a$. Second, use gradient descent on $x_a$ to make $L_{\text{CE}}(F(x_a), l)$ bigger and the change for each pixel is $\leq 0.i$.

Figure 5: Three types adversaries (left-side) and their outputs from $F$ (middle) are given in the three rows. The right-side: output of $D(g(E(x), l_x))$, $l_x$ is the label of $x$.

We will compare with the well-known adversarial training method proposed in \cite{22} for a CNN whose structure is given in Appendix B. The results in Table 3 are the adversarial creation rates from the test set. The adversarial creation rate for type 1.10 adversaries by $E$ is about 35%. We use 1000 type 1 adversaries as input to $F$. Among the 1000 inputs, 733 samples are considered as problem images, 267 samples are given wrong labels. So, for about 26.7% of the type 1 adversaries of $E$, $F$ gives wrong labels and 73.3% of them are considered as problem images. Therefore, the adversary creation rate for type 1 adversaries is $9.35\% = 35\% \cdot 26.7\%$. Results for other types are computed similarly.

| Attack   | AT  | CAE | LCAE | NLabels |
|----------|-----|-----|------|---------|
| Type 1.10| 6\% | 3\% | < 1\%| 3.04    |
| Type 1.20| 15\%| 15\%| < 1\%| 3.10    |
| Type 1.30| 55\%| 22\%| 1\%  | 3.13    |
| Type 2.40| 55\%| 19\%| < 1\%| 3.58    |
| Type 2.60| 76\%| 23\%| 1\%  | 3.64    |
| Type 2.80| 85\%| 30\%| 2\%  | 3.70    |
| Type 3.0 | 2\% | 1\% | < 1\%| 2.87    |
| Type 3.1 | 18\%| 11\%| 1\%  | 3.22    |
| Type 3.2 | 51\%| 34\%| 4\%  | 3.94    |

Table 3: Adversarial creation rates for CAE (second column) and LCAE (third column; see section 4.3). Attack is the attack methods: we use PGD-$i$ for type 1.i adversaries and JSMA for type 2.i adversaries. AT is the results for the network trained with adversarial training \cite{22}.

From Table 3 we have the following observations. (1) CAE is always better than the adversarial training, and much better for more difficult adversaries. The performance of CAE is more stable than that the adversarial training, whose adversarial creation rates are less than 30\%. For adversarial training, the adversarial creation rate for type 2.80 attack method is 85\%. (2) The results for $F$ and the adversarial training are different. Since the input samples are adversaries of $E$, the CAE cannot give the correct label for these adversaries and the adversarial creation rate for CAE is the percentage of inputs for which CAE give wrong labels. We can see that CAE also performs better for strong adversaries.
4 List classifier to defend adversaries

It was widely believed that adversaries are inevitable for the current DNN framework [2, 5, 29]. A possible way to alleviate the problem is to give several labels instead of one. In this section, we give such an approach based on the CAE.

4.1 The list classifier LCAE

The list classification algorithm is motivated by Figure 5, where the label $L(x)$ is wrong, but $D(g(E(x), l_x))$ gives a very close approximation to the input $x$. The idea is to output all labels $l$ such that $D(g(E(x), l))$ is similar to $x$ with “high probabilities”.

We first define a distance between two images. Let $x \in \mathbb{R}^n$, and $A(x)$ the average of all coordinates of $x$. For $a \in \mathbb{R}$, define $S(a) = 1$ if $a > 0$, and $S(a) = 0$ if $a \leq 0$. Define $\hat{x} = (\hat{x}_1, \ldots, \hat{x}_n) \in \{0, 1\}^n$, where $\hat{x}_i = S(x_i - A(x))$. We call $\hat{x}$ the standardization of $x$. Define the distance between $x$ and $y$ as $\text{Dis}(x, y) = \frac{A(x - y)}{A(x + y)}$ for $x, y \in \mathbb{R}^n$.

Suppose that $F$ in (1) is a trained CAE with $L$ as the label set. We do image classification by giving several possible answers.

Algorithm 2 LCAE

The input: $x \in \mathbb{R}^n$, hyperparameters $B$ and $D$ (in the experiments below, we choose $B=14$ and $D=10\%$). The output: a list of labels and the corresponding images.

S1 Compute $F(x)$ with (5) and $L(x)$ with (8).

S2 Let $x_l = D(g(E(x), l))$, for all $l \in L$.

S3 Compute $L_l = ||x_l - x||$, $D_l = \text{Dis}(\hat{x}_l, \hat{x})$, and $E_l = D_l L_l$ for $l \in L$.

S4 The probability for $x$ to be an outlier is $P_{-1} = \frac{1 - e^{-(B E_m)^2}}{1 + e^{-(B E_m)^2}}$, where $E_m = \min_l E_l$.

S5 The probability for $x$ to have label $l$ is $P_l = (1 - P_{-1}) \tilde{P}_l$, where $\tilde{P}_l = \frac{1/E_l}{\sum_j 1/E_j}$ and $l \in L$.

S6 Output $(l, x_l)$ if $P_l > D$ for $l \in L$.

We give an illustrative example. In Figure 6 the images $x, x_l, \hat{x}, \hat{x}_l$ are given. In Table 4 the probabilities $P_l$ are given. The probability of the picture to have label 1 is 88.80% and all other probabilities are smaller than 0.1.

![Figure 6: The first row is x and x_l in S2 of Algorithm 2 and the second row is \(\hat{x}\) and \(\hat{x}_l\) for \(l = 0, \ldots, 9\).](image)
| $l$ | $-1$ | $0$ | $1$ | $2$ | $3$ | $4$ | $5$ | $6$ | $7$ | $8$ | $9$ |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| $\text{Dis}(x, x_l)$ | 0.0473 | 0.0029 | 0.0290 | 0.0727 | 0.0266 | 0.0727 | 0.0515 | 0.0266 | 0.0727 | 0.0515 | 0.0266 |
| $\text{Dis}(\bar{x}, \bar{x}_l)$ | 0.3848 | 0.069 | 0.2759 | 0.0727 | 0.0282 | 1 | 0.4951 | 0.2902 | 0.2773 | 1 |
| $E_l$ | 0.0182 | 0.0002 | 0.0080 | 0.0727 | 0.0075 | 0.0727 | 0.0255 | 0.0074 | 0.0099 | 0.0727 |
| $\bar{P}_l$ | 0.0098 | 0.8881 | 0.0222 | 0.0024 | 0.0237 | 0.0024 | 0.0070 | 0.0240 | 0.0179 | 0.0024 |
| $P_l$ | 0.01% | 0.98% | 88.80% | 2.22% | 0.24% | 2.37% | 0.24% | 0.70% | 2.40% | 1.79% | 0.24% |

Table 4: The probabilities for $x$ to have label $l$

In the rest of this section, we give experimental results for LCAE. The structure and hyperparameters of the CAE are the same as that used in the preceding section. The codes can be found at https://github.com/yyyylllj/NetB.

### 4.2 Accuracy on the test set

We use LCAE to the test set of MNIST, which contains 10000 samples. The number of samples whose output contains the correct label is 9997. The number of samples whose output only contains the correct label is 5712. The number of samples whose output contains outlier is 1. In Table 5, we give the numbers of labels in the output lists. The average number of labels in the output list is 1.722.

Comparing to the result in Table 1, at the cost of outputting a list containing 1.722 labels, the accuracy could be increased from 98.64% to 99.97%.

| Number of labels | 1 | 2 | 3 | 4 | 5 | 6 |
|-----------------|---|---|---|---|---|---|
| Number of samples | 5712 | 2122 | 1500 | 560 | 104 | 2 |

Table 5: Number of labels in the output

In Figure 7, we give the unique sample whose output contains the outlier. In Figures 8, 9, 10, we give the three samples whose outputs do not contain the correct label. In Table 6, we give the probabilities of these figures.

Figure 7: Output contains the correct label 0 and the outlier. The first row is $x$ and $x_l$ in S2 of Algorithm 2 and the second row is $\bar{x}$ and $\bar{x}_l$, for $l = 0, \ldots, 9$.

Figure 8: Output contains labels 0, 3, 6, but does not contain the correct label 5.

Figure 9: Output contains labels 3 and 9, but does not contain the correct label 5.
Table 6: The probabilities for the inputs in Figures 7, 8, 9, 10

| $l = \ldots$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|--------------|---|---|---|---|---|---|---|---|---|---|
| Figure 7     | $P_l$ | 12.5% | 13.1% | 11.7% | 12.9% | 4.52% | 4.20% | 4.20% | 9.39% | 8.50% | 11.7% | 7.21% |
| Figure 8     | $P_l$ | 0.25% | 36.2% | 1.91% | 6.94% | 14.7% | 1.25% | 8.27% | 12.4% | 1.34% | 12.5% | 4.08% |
| Figure 9     | $P_l$ | 0.01% | 0.63% | 0.75% | 0.43% | 59.5% | 0.43% | 4.68% | 0.43% | 0.65% | 2.64% | 29.7% |
| Figure 10    | $P_l$ | 0.01% | 0.82% | 0.96% | 0.77% | 76.6% | 0.53% | 7.42% | 0.53% | 0.56% | 4.95% | 6.77% |

4.3 Defend adversaries

We check the ability of LCAE to defend adversaries for the three types of adversaries given in section 3.3. The creation rates of adversaries for LCAE are given in the column “LCAE” of Table 3 and the numbers of labels in the outputs of LCAE are given in column “NLabels”. From these tables, we can see that the classification list almost always contains the correct labels for various adversaries by outputting about 3.5 labels. In summary, the list classification can be considered to give an uncertain but lossless classification for these adversaries at the costs of giving about 3.5 labels from Table 3.

In the following figures, we give three adversaries and the outputs of LCAE. In Table 7 we give the probabilities for these adversaries.

Figure 11: A type 1.10 adversary of 7 (section 3.3); the correct label 7 is given. The first row is $x$ and $x_l$ in S2 of Algorithm 2 and the second row is $\hat{x}$ and $\hat{x}_l$, for $l = 0, \ldots, 9$.

Figure 12: A type 2.80 adversary of 5 (section 3.3); the wrong output label is 8.

Figure 13: A type 3.1 adversary of 7 (section 3.3); the wrong output label is 8.
Table 7: Probabilities for the samples in Figures 11, 12, 13.

4.4 Recognize outliers

In section 4.3, it has already shown that CAE can recognize almost all outliers. In this section, we give a more detailed analysis on the ability of CAE to defend outliers by giving the results of LCAE. In Table 8, we give the percentages of the samples whose $P_{-1}$ is larger than 50% and 80%, respectively. From the table, we see that $P_{-1} > 50\%$ for all images. If a sample has $P_{-1} > 50\%$, it is certain to be an outlier and this explains the results in Table 2 in more detail.

| Outlier | $P_{-1} > 50\%$ | $P_{-1} > 80\%$ |
|---------|----------------|----------------|
| Type 1  | 100\%          | 100\%          |
| Type 2  | 100\%          | 100\%          |
| Type 3  | 100\%          | 64\%           |
| Type 4  | 100\%          | 98\%           |

Table 8: The percentages of outliers which have probability bigger than 50% and 80%

In Figures 14 and 15, two outliers in Figure 3 are given, respectively. Their corresponding probabilities are given in Table 9.

Figure 14: A type 3 outlier from Figure 3 and the output from LCAE.

Figure 15: A type 4 outlier in Figure 3 and the output from LCAE.

Table 9: The probabilities for the images in Figures 14 and 15

4.5 Decouple-classification with LCAE

Consider a special kind of outliers obtained by “adding” two images from MNIST with different labels, some of which are given in Figure 16. These images are not numbers, but they have strong characteristic of numbers.
As an application of LCAE, we give a decouple-classification algorithm which can be used to find one or more of the elements in a sample containing two or more well-mixed elements from $\mathcal{O}$.

We use 1000 outliers of this kind as inputs to LCAE. For about 54% of these outliers, the two numbers used to form the images are found by LCAE, and for 99% of them, one of the two numbers is found.

We give examples in Figures 17 and 18 and the corresponding probabilities are given in Table 10.

**Figure 17:** LCAE finds the two numbers 2 and 9 used to form the image.

**Figure 18:** LCAE finds one of the two numbers 4 and 5 used to form the image.

| Figure | $l$ | $t$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|--------|-----|-----|---|---|---|---|---|---|---|---|---|
| 17     | $P_t$ | 3.07% | 3.25% | 7.39% | 20.4% | 11.4% | 12.2% | 1.73% | 1.73% | 2.14% | 20.4% |
| 18     | $P_t$ | 2.50% | 3.31% | 3.99% | 9.02% | 10.0% | 12.3% | 4.44% | 1.94% | 6.83% | 21.9% |

Table 10: The probabilities for the images in Figures 17 and 18.

5 Conclusion

In this paper, we consider the problem of building robust DNNs to defend outliers and adversaries. In the open-world classification problem, the inputs to the DNN could be outliers which are irrelevant to the training set. The objects to be classified usually consist of a low-dimensional subspace of the total input space to the DNN and the majority of the input are outliers. So, the DNN need to recognize outliers in order to be used in open-world applications. A more subtle robust problem is that, for almost all DNNs with moderate complex structures, there exist adversary samples. The ability to defend adversaries is important for the DNN to be used in safety-critical applications.

In this paper, we present a new neural network structure which is more robust to recognize outliers and to defend adversaries. The basic idea is to change the autoencoder from an un-supervised learning method to a classifier, where the encoder maps images with different labels into disjoint compression subspaces and the decoder recovers the image from its compression subspace.

The newly introduced classification-autoencoder can recognize almost all outliers due to the fact that the output of the autoencoder is always similar the objects to be classified, and hence achieves state of the art for outlier recognition.
Since adversaries are seemingly inevitable for the current DNNs, we introduce the list-classification based on the CAE, which outputs several labels instead of one. According to our experiments, the list classifier can give near lossless classification in the sense that the output list contains the correct label for almost all adversaries and the size of the output list is reasonably small.

An overall framework for robust classification could be done as follows. First, CAE and LCAE are used to identify those inputs which are elements of $\mathcal{O}$ or outliers with high probabilities. Second, for the remaining “fuzzy” samples, we may use LCAE to output several labels and their corresponding recovered images for further analysis.

References

[1] N. Akhtar and A. Mian. Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey. arXiv:1801.00553v3, 2018.

[2] A. Azulay and Y. Weiss. Why Do Deep Convolutional Networks Generalize so Poorly to Small Image Transformations. Journal of Machine Learning Research, 20, 1-25, 2019.

[3] T. Bai, J. Luo, J. Zhao. Recent Advances in Understanding Adversarial Robustness of Deep Neural Networks. arXiv:2011.01539, 2020.

[4] D.H. Ballard. Modular Learning in Neural Networks. Proc. AAAI’87, Vol. 1, 279-284, AAAI Press, 1987.

[5] A. Bastounis, A.C. Hansen, V. Vlačić. The Mathematics of Adversarial Attacks in AI - Why Deep Learning is Unstable Despite the Existence of Stable Neural Networks, arXiv:2109.06098, 2021.

[6] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, F. Roli. Evasion Attacks Against Machine Learning at Test Time. Proc. of European Conference on Machine Learning and Knowledge Discovery in Databases, 387-402, Springer, 2013.

[7] J. Buckman, A. Roy, C. Raffel, I. Goodfellow. Thermometer Encoding: One Hot Way to Resist Adversarial examples. Proc. of the 6th International Conference on Learning Representations, Vancouver, Canada, 2018.

[8] R. Chalapathy and S. Chawla. Deep Learning for Anomaly Detection: A Survey. arXiv:1901.03407v2, 2019.

[9] C.Q. Choi. 7 Revealing Ways AIs Fail. IEEE Spectrum, 42-47, October, 2021.

[10] M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, N. Usunier. Parseval Networks: Improving Robustness to Adversarial Examples. Proc. ICML’2017, 854-863, 2017.

[11] G. Cybenko. Approximation by Superpositions of a Sigmoidal Function. Mathematics of Control, Signals and Systems, 2(4): 303-314, 1989.

[12] I.J. Goodfellow, Y. Bengio, A. Courville. Deep Learning, MIT Press, 2016.

[13] I.J. Goodfellow, J. Shlens, C. Szegedy. Explaining and Harnessing Adversarial Examples. arXiv:1412.6572, 2014.
[14] C. Guo, M. Rana, M. Cisse, L. van der Maaten. Countering Adversarial Images using Input Transformations. ArXiv: 1711.00117, 2017.

[15] M. Hein, M. Andriushchenko. Formal Guarantees on the Robustness of a Classifier Against Adversarial Manipulation. Proc. NIPS, 2266-2276, 2017.

[16] G. Hinton, O. Vinyals, J. Dean. Distilling the Knowledge in a Neural Network. arXiv:1503.02531, 2015.

[17] K. Hornik. Approximation Capabilities of Multilayer Feedforward Networks. Neural Networks, 4(2): 251-257, 1991.

[18] Y. LeCun, Y. Bengio, G. Hinton. Deep Learning. Nature, 521(7553), 436-444, 2015.

[19] N. Lei, D. An, Y. Guo, K. Su, S. Liu, Z. Luo, Z. Gu. A Geometric Understanding of Deep Learning. Engineering, 6(3), 361-374, 2020.

[20] M. Leshno, V.Ya. Lin, A. Pinkus, and S. Schocken. Multilayer Feedforward Networks with a Non-polynomial Activation Function Can Approximate any Function. Neural Networks, 6(6): 861-867, 1993.

[21] W. Lin, Z. Yang, X. Chen, Q. Zhao, X. Li, Z. Liu, J. He. Robustness Verification of Classification Deep Neural Networks via Linear Programming. CVPR’2019, 11418-11427, 2019.

[22] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. arXiv:1706.06083, 2017.

[23] A. Okabe, B. Boots, K. Sugihara, S.N. Chiu, D.G. Kendall. Okabe, Michiko, Barry Boots, and Sung Nok Chiu. Spatial Tessellations: Concepts and Applications of Voronoi Diagrams. John Wiley & Sons, New York, 2000.

[24] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z.B. Celik, A. Swami, The Limitations of Deep Learning in Adversarial Settings. In 2016 IEEE European Symposium on Security and Privacy, 372-387, IEEE Press, 2016.

[25] A. Pinkus. Approximation Theory of the MLP Model in Neural Networks. Acta Numerica, 8: 143-195, 1999.

[26] Y. Qi, Y. Wang, X. Zheng, Z. Wu. Robust Feature Learning by Stacked Autoencoder with Maximum Correntropy Criterion. ICASSP’2014, 6716-6720, IEEE Press, 2014.

[27] A. Raghunathan, J. Steinhardt, P. Liang. Certified Defenses Against Adversarial Examples. ArXiv: 1801.09344, 2018.

[28] A.V.H. Rasmus, M. Honkala, M. Berglund, T. Raiko. Semi-supervised Learning with Ladder Networks. NIPS’15, 3546–3554, 2015.

[29] A. Shafahi, W.R. Huang, C. Studer, S. Feizi, T. Goldstein. Are Adversarial Examples Inevitable? arXiv:1809.02104, 2018.

[30] A. Shafahi, M. Najibi, A. Ghiasi, Z. Xu, J. Dickerson, C. Studer, L.S. Davis, G. Taylor, T. Goldstein. Adversarial Training for Free! ArXiv: 1904.12843, 2019.
[31] P. Samangouei, M. Kabkab, R. Chellappa. Defense-GAN: Protecting Classifiers Against Adversarial Attacks using Generative Models. ArXiv: 1805.06605, 2018.

[32] W.J. Scheirer, A. Rocha, A. Sapkota, T.E. Boult. Towards Open Set Recognition. IEEE Trans. PAMI, 36(7):1757-1772, 2013.

[33] Y. Song, T. Kim, S. Nowozin, S. Ermon, N. Kushman. Pixeldefend: Leveraging Generative Models to Understand and Defend Against Adversarial Examples. ArXiv: 1710.10766, 2017.

[34] M.H. Stone. The Generalized Weierstrass Approximation Theorem. Mathematics Magazine, 21(4): 167-184, 1948.

[35] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I.J. Goodfellow, R. Fergus. Intriguing Properties of Neural Networks, arXiv:1312.6199, 2013.

[36] F. Tramer, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, P. McDaniel. Ensemble Adversarial Training: Attacks and Defenses. ArXiv: 1705.07204, 2017.

[37] P. Vincent, H. Larochelle, Y. Bengio, P.A. Manzagol. Extracting and Composing Robust Features with Denoising Autoencoders. Proc. ICML’08, 1096-1103, ACM Press, 2008.

[38] C.H. Xie, J.Y. Wang, Z.S. Zhang, Z. Ren, A. Yuille. Mitigating Adversarial Effects Through Randomization. ArXiv: 1711.01991, 2017.

[39] E. Wong, J. Z. Kolter. Provable Defenses Against Adversarial Examples via the Convex Outer Adversarial Polytope. ArXiv: 1711.00851, 2017.

[40] H. Xu, Y. Ma, H.C. Liu, D. Deb, H. Liu J.L. Tang, A.K. Jain. Adversarial Attacks and Defenses in Images, Graphs and Text: A Review. International Journal of Automation and Computing, 17(2), 151-178, 2020.

[41] Z. Yang, X. Wang, Y. Zheng. Sparse Deep Neural Networks Using $L_{1,\infty}$-Weight Normalization. Statistica Sinica, 2020, doi:10.5705/ss.202018.0468.

[42] L. Yu and X. S. Gao. Improve the Robustness and Accuracy of Deep Neural Network with $L_{2,\infty}$ Normalization. arXiv:2010.04912.

[43] L. Yu and X. S. Gao. Robust and Information-theoretically Safe Bias Classifier against Adversarial Attacks. arXiv:2111.04404, 2021.

[44] D.H. Zhang, T.Y. Zhang, Y.P. Lu, Z.X. Zhu, B. Dong. You Only Propagate Once: Accelerating Adversarial Training via Maximal Principle. ArXiv: 1905.00877, 2019.

[45] X.Y. Zhang, C.L. Liu, C.Y. Suen. Towards Robust Pattern Recognition: A Review. Proc. of the IEEE, 108(6), 894-922, 2020.

[46] C. Zhou and R.C. Paffenroth. Anomaly Detection with Robust Deep Autoencoders. KDD’17, 665-674, ACM Press, 2017.
Appendix A. Proof of Theorem 1

First introduce the notion of Voronoi tessellation. Let \( I_0 = (0,1) \subset \mathbb{R} \), \( A \subset I_0^n \) a convex open set, and \( P = \{ p_i, i = 1, \ldots, t \} \subset A \). For each \( p_i \), let \( R_i \) be the set of points in \( A \) which are strictly closer to \( p_i \) than to \( p_j, j \neq i \). Then \( R_i \) is an open convex set and \( \{ R_i \}_{i=1}^t \) are called the Voronoi tessellation of \( A \) generated by \( P \), and \( R_i \) is called the Voronoi region with generating point \( p_i \). It is clear that \( A = \bigcup_{i=1}^t (R_i) \), where \( \overline{R}_i \) is the closure of \( R_i \).

We first consider the case where the objects to be classified consist of a single connected open set.

**Lemma 5.1.** Let \( S \) be a bounded open set in \( \mathbb{R}^m \) and \( A \) a bounded and convex open set in \( \mathbb{R}^m \). For any \( \varepsilon, \gamma \in \mathbb{R}_+ \) and any \( m \in \mathbb{N}_+ \), there exist functions \( \text{E} : I_0^m \to \mathbb{R}^m \) and \( D : \mathbb{R}^m \to I_0^m \), which are piecewise continuous functions with a finite number of continuous regions and satisfy

0. \( \text{E}(x) = 0 \) if \( x \notin S \) and \( D(y) = 0 \) if \( y \notin A \).

1. \( V_1 = \{ x \in S \subset I_0^m \mid \text{E}(x) \in A \} \) satisfies \( \frac{V(V_1)}{V(S)} > 1 - \varepsilon \);

2. \( V_2 = \{ y \in A \subset \mathbb{R}^m \mid D(y) \in S \} \) satisfies \( \frac{V(V_2)}{V(A)} > 1 - \varepsilon \);

3. \( V_3 = \{ x \in S \subset I_0^m \mid \| x - D(E(x)) \| < \gamma \} \) satisfies \( \frac{V(V_3)}{V(S)} > 1 - \varepsilon \).

**Proof.** Without loss of generality, assume \( S \subset I_0^n \). Let \( k \in \mathbb{N}_+ \) and

\[
C^{k,d_1,d_2,\ldots,d_n} = \left( \frac{d_1}{k}, \frac{d_1+1}{k} \right) \times \left( \frac{d_2}{k}, \frac{d_2+1}{k} \right) \cdots \times \left( \frac{d_n}{k}, \frac{d_n+1}{k} \right),
\]

where \( d_i \in \{0,1,2,\ldots,k-1\} \). Let

\[
S_k = \{ C^{k,d_1,d_2,\ldots,d_n} \mid C^{k,d_1,d_2,\ldots,d_n} \subset S \}.
\]

It is easy to see that \( V(S_k) = \frac{t_k}{k^n} \), where \( t_k \) is the number of cubes in \( S_k \). When \( k \) becomes lager, \( V(S_k) \) will increase and approach to \( V(S) \). Then, we can choose a \( k \) such that

\[
k > \frac{\sqrt{m}}{\gamma} \quad \text{and} \quad \frac{V(S_k)}{V(S)} = \frac{t_k}{t_m} > 1 - \varepsilon. \quad (10)
\]

For simplicity, let \( S_k = \{ C_i \}_{i=1}^{t_k} \). Let

\[
A_k = \{ a_i \}_{i=1}^{t_k}
\]

be \( t_k \) distinct points in \( A \). Let \( R_i = \{ R_i \}_{i=1}^{t_k} \) be the Voronoi tessellation of \( A \) generated by \( A_k \) and \( a_i \) the generating point for \( R_i \). Define \( E \) and \( D \) as follows

\[
E(x) : I_0^m \to \mathbb{R}^m, E(x) = a_i \quad \text{if} \quad x \in C_i \quad \text{and} \quad E(x) = 0 \quad \text{otherwise.} \quad (12)
\]

\[
D(y) : \mathbb{R}^m \to I_0^m, D(y) = c_i \quad \text{if} \quad y \in R_i
\]

where \( c_i \) is the center of \( C_i \). It is clear that \( E(x) \) is a constant function over each \( C_i \) and \( D(x) \) is a constant function over each \( R_i \).

We now prove that \( E \) and \( D \) satisfy the properties of the lemma. From the above construction, we have \( V_1 = \bigcup_{i=1}^{t_k} C_i \) and \( V(V_1) = V(S_k) \). Then property 1 follows from (10). It is easy to see \( V_2 = \bigcup_{i=1}^{t_k} R_i \) and \( V_2 = A \). Then \( \frac{V(V_2)}{V(A)} = 1 \), and property 2 is proved. For \( C_i \subset S_k \), if \( x \in C_i \), then \( D(E(x)) \) is the center of \( C_i \), and hence \( \| D(E(x)) - x \| < \frac{\sqrt{m}}{k} < \gamma \) by (10). Then \( V_3 = V_1 \) and the lemma is proved.  \( \square \)
From the proof of Lemma 5.1, we have

**Corollary 5.1.** Use the notations in Lemma 5.1. There exists a number $t$ such that

1. $V_1 = V_2 = \bigcup_{i=1}^t C_i$, where $C_i \subset S$ are disjoint open cubes with center $c_i$.
2. Let $A_t = \{a_i\}_{i=1}^t$ be $t$ distinct points inside $A$ and $R_t$, $i = 1, \ldots, t$ the Voronoi polyhedra generated by $A_t$ in $A$ and $a_i \in R_t$. Then $V_2 = \bigcup_{i=1}^t R_t$ and $A = V_2$.

3. $E(x) = a_i$ for $x \in C_i$ and $E(x) = 0$ otherwise. $D(y) = c_i$ for $y \in R_t$ and $D(y) = 0$ otherwise.

For a set $W \subset \mathbb{R}^n$ and $a \in \mathbb{R}_+$, define $W^a = \{x \in W \mid \exists r > a, \text{s.t. } B(x, r) \subset W\}$.

**Lemma 5.2.** Let $F : \mathbb{R}^n_0 \to \mathbb{R}^m$ be a piecewise linear function with a finite number of linear regions. Then for any $\epsilon > 0$, there exists a DNN $F : \mathbb{R}^n_0 \to \mathbb{R}^m$ such that $|F(x) - F(x')| < \epsilon$ for $x \in A'$, where $A$ is any linear region of $F$. Moreover, $D = \{x \mid |F(x) - F(x')| > \epsilon\}$ satisfies $V(D) < \alpha$.

**Proof.** It is easy to see that there exists a continuous function $H$ which satisfies that $|F(x) - H(x)| = 0$ for $x \in A'$. Then the lemma follows from the universal approximation theorem of DNN [17, 20, 25].

Now we give the proof of Theorem 1.

**Proof.** Let $A_0 = \mathbb{R}^n_0$ and $A_j = \{y \mid y = z + 2j\mathbf{1}, z \in A_{j-1}\}$ for $j = 1, \ldots, o$, where $1 \in \mathbb{R}^m$ is the vector whose coordinates are 1. We first assume that each $S_I$ is connected. By Lemma 5.1 and Corollary 5.1, for each $l \in \mathbb{L}$ and $e_1, e_2 \in \mathbb{R}_+$, there exist functions $E_I : \mathbb{R}^n \to \mathbb{R}^m$ and $D_I : \mathbb{R}^m \to \mathbb{R}^n$, which are piecewise continuous functions with a finite number of continuous regions and satisfy

(A1) $V_{l,1} = \bigcup_{i=1}^t C_{l,i}$, where $C_{l,i} \subset S_l$ are disjoint open cubes with center $c_{l,i}$. Furthermore, $\frac{V(V_{l,1})}{V(S_l)} > 1 - \epsilon_1$ and $|x - D_l(E_l(x))| < \gamma$ for $x \in V_{l,1}$.

(A2) $U_l = \{a_{l,i}\}_{i=1}^t$ is a set of $t$ distinct points inside $A_l$ and $R_{l,i}, i = 1, \ldots, t$ the Voronoi polyhedra generated by $U_l$ inside $A_l$ and $a_{l,i} \in R_{l,i}$. Therefore, $A_l = \bigcup_{i=1}^t R_{l,i}$.

(A3) $E_l(x)$ and $D_l(y)$ are defined as follows: $E_l(x) = a_{l,i}$ for $x \in C_{l,i}$ and $E_l(x) = 0$ otherwise; $D_l(y) = c_{l,i}$ for $y \in R_{l,i}$ and $D_l(y) = 0$ otherwise.

Define $E$ and $D$ as follows: $E(x) = \sum_{i=0}^o E_i(x)$ and $D(y) = \sum_{i=0}^o D_i(y)$. Note that $E$ is constant over $C_{l,i}$ and $D$ is constant over $R_{l,i}$. We call $C_{l,i}$ and $R_{l,i}$ constant regions of $E$ and $D$, respectively.

By Lemma 5.2, for any $\epsilon_2, \epsilon_3$, and $\beta$, there exist DNNs $E$ and $D$ such that

(B1) $|E(x) - E(x')| < \epsilon_2$, for $x \in \mathbb{R}^n_0 \setminus S_{-1}$, where $S_{-1} \subset \mathbb{R}^n_0$, $V(S_{-1}) < \epsilon_3$, and $C_{l,i}^{\beta/4} \cap S_{-1} = \emptyset$ for any constant region $C_{l,i}$ of $E$.

(B2) $|D(y) - D(y')| < \epsilon_2$, for $y \in A \setminus S_{-2}$, where $A = \bigcup_{i=0}^o A_i$, $S_{-2} \subset A$, $V(S_{-2}) < \epsilon_3$, and $R_{l,i}^{\beta/4} \cap S_{-2} = \emptyset$ for any constant region $R_{l,i}$ of $D$.

Let $\beta_l$ be the minimum of the distances between any pair of points in $E(S_l) = U_l = \{a_{l,i}\}_{i=1}^t$ and the distances between any point in $E(S_l)$ and the surface of $A_l$. Choose the parameters such that

(C0) $\beta < \beta_l$ for all $l \in \mathbb{L}$.
(C1) $\epsilon_2 \leq \beta/4$ and $\epsilon_2 \leq 1/2$.

(C2) $\epsilon_3 < \epsilon \sum_i V(S_i)$.

(C3) $\gamma + \epsilon_2 < \gamma$, $\epsilon_1 + \frac{\epsilon_2}{V(S_i)} < \epsilon$ for all $l$.

We now prove that $E$ and $D$ satisfy the properties in the theorem. Let $x \in K$. Then $E(x) \in E(S_i) \cap E(S_j)$ for $i \neq j$. By property (B1), if $x \in S_i/S_{-1}$ then $E(x) \in D_i = (-\epsilon_2, 1 + \epsilon_2)^m + 2i1$. Similarly, if $x \in S_j/S_{-1}$ then $E(x) \in D_j = (-\epsilon_2, 1 + \epsilon_2)^m + 2j1$. By (C1), we have $D_i \cap D_j = \emptyset$ if $i \neq j$. So $x \in K$ implies $x \in S_{-1}$, and hence $V(K) < \epsilon_3$ by (B1). Because of (C2), we have $\frac{V(K)}{\sum_i V(S_i)} < \epsilon$. The first property of the theorem is proved.

Let $x \in C_{t,i}/S_{-1}$. We will prove

$$||D(E(x)) - x|| \leq ||D(E(x)) - x|| + \epsilon_2 \leq \gamma + \epsilon_2 \leq \gamma. \quad (13)$$

The first inequality in (13) follows from $||D(E(x)) - D(E(x))|| < \epsilon_2$ which will be proved below. By (A3), $E(x) = a_{l,i} \in R_{t,i}$ for $x \in C_{l,i}$. For any $x_0 \in B(E(x), \beta/2)$, $E(x)$ is the nearest point of $x_0$ in $E(S_i)$ by (C0), and hence $B(E(x), \beta/2) \subset R_{t,i}$ is a constant region of $D(y)$ by (A2). By (C1), $B(E(x), \epsilon_2) \subset B(E(x), \beta/2) \subset R_{t,i}$ and hence $B(E(x), \epsilon_2) \subset R_{t,i}^{\beta/4}$. By (B1), $E(x) \in B(E(x), \epsilon_2)$ since $x \in C_{t,i}/S_{-1}$, hence $E(x) \in R_{t,i}^{\beta/4}$. By (B2), $E(x) \notin S_{-2}$ and hence $||D(E(x)) - D(E(x))|| < \epsilon_2$. The first inequality in (13) is proved.

We now prove the second inequality in (13). We already proved $E(x) \in B(E(x), \epsilon_2) \subset R_{t,i}$ and hence $D(E(x)) = D(E(x))$ by (A3). By (A1), $||D(E(x)) - x|| = ||D(E(x)) - x|| < \gamma$ since $x \in C_{l,i} \subset V_{l,i}$. The last inequality in (13) follows from (C3).

By (13), $V_{t,i}/S_{-1} \subset \widehat{S}_t$. Finally, by (B1), (A1), and (C3), we have, $\frac{V(\widehat{S}_t)}{V(S_t)} \geq \frac{V(V_{t,i}/S_{-1})}{V(S_t)} > \frac{V(V_{t,i})-\epsilon_3}{V(S_t)} > (1 - \epsilon_1) - \frac{\epsilon_2}{V(S_t)} > 1 - \epsilon$. The theorem is proved.

In the above proof, each $S_i$ is assumed to be connected. It is clear that the proof can be easily modified for the case such that different $S_i$ have the same label.

### Appendix B. Structure of the network

We give the structure of the networks used in the experiments. We first give the structure of the CAE.

**The structure of $E$:**

Input layer: $N \times 1 \times 28 \times 28$, where $N$ is steps of training.

Hidden layer 1: a convolution layer with kernel $1 \times 10 \times 3 \times 3$ with padding= 1 → do a batch normalization → do Relu → use max pooling with step=2.

Hidden layer 2: a convolution layer with kernel $10 \times 28 \times 3 \times 3$ with padding= 1 → do a batch normalization → do Relu → use max pooling with step=2.

Hidden layer 3: a convolution layer with kernel $28 \times 28 \times 3 \times 3$ with padding= 1 → do a batch normalization → do Relu → use max pooling with step=2.

Hidden layer 4: draw the output as $N \times 252$ → use a full connection with output size $N \times 168$ → do Relu.

Output layer: a full connection layer with output size $N \times 100$ → do Relu.

**The structure of $D$:**
Input layer: $N \times 100$
Hidden layer 1: a full connection layer with output size $N \times 252$ → do Relu.
Hidden layer 2: a full connection layer with output size $N \times 600$ → do Relu.
Hidden layer 3: a full connection layer with output size $N \times 2700$ → do Relu.
Hidden layer 4: draw the output as $N \times 3 \times 30 \times 30$ → use convolution with kernel $3 \times 28 \times 3 \times 3$ with padding= 0 → do a batch normalization → do Relu.
Output layer: use convolution with kernel $28 \times 1 \times 1 \times 1$ with padding= 0 → do Relu.

In section 3.2 a CNN $H$ is used to recognize outliers. In section 3.3 a CNN trained with adversarial training is used for comparison. These two CNNs have the following structure.

**Input layer**: $N \times 1 \times 28 \times 28$.

Hidden layer 1: a convolution layer with kernel $1 \times 32 \times 3 \times 3$ with padding= 1 → do Relu → use max pooling with step=2.

Hidden layer 2: a convolution layer with kernel $32 \times 64 \times 3 \times 3$ with padding= 1 → do Relu → use max pooling with step=2.

Hidden layer 3: a convolution layer with kernel $64 \times 64 \times 3 \times 3$ with padding= 1 → do Relu → use max pooling with step=2.

Hidden layer 4: draw the output as $N \times 576$ → use a full connection layer with output size $N \times 128$ → do Relu.

**Output layer**: a full connection layer with output size $N \times 10$. The output layer of $H$ is a full connection layer with output size $N \times 11$.