Information Sciences 550 (2021) 57–70

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A robust generative classifier against transfer attacks based on variational auto-encoders ☆

Chen Zhang [a,b], Zhuo Tang [a,b,*], Youfei Zuo [a,b], Kenli Li [a,b], Keqin Li [a,c]

[a] College of Computer Science and Electronic Engineering, Hunan University, Changsha, China
[b] National Supercomputing Center in Changsha, China
[c] Department of Computer Science, State University of New York, New York, USA

ARTICLE INFO

Article history:
Received 4 May 2020
Received in revised form 10 October 2020
Accepted 16 October 2020
Available online 1 November 2020

Keywords:
Adversarial examples
Robustness
Transfer attacks
VAEs

ABSTRACT

Deep neural networks (DNNs) are vulnerable to adversarial examples. Even under black-box setting that is without access to the target model, transfer-based attacks can easily fool the DNNs. To alleviate this problem, we propose a robust classification model against transfer attacks based on the framework of variational Auto-Encoders (VAEs) which are probabilistic generative models and have been successfully used to a large mount of tasks. Specifically, our model simulates the data generative process with several multivariate Gaussian distributions and DNNs: (1) We assume that the latent embedding generated by an *encoder* (a DNN) of each category corresponds to a multivariate Gaussian distribution. (2) A *decoder* (a DNN) is proposed to decodes the latent embedding into an observable. (3) Theoretical analysis illustrates that our model can predict data's labels by maximizing the lower bound on the log-likelihood for each category utilizing Bayes' theorem with excellent robustness against transfer attacks. Inference in our model is done in a variational way so the Stochastic Gradient Variational Bayes (SGVB) estimator and reparamerization trick can be utilized to optimize the evidence lower bound (ELBO). The experiments with quantitative comparisons show that our approach reaches state-of-the-art with significantly better robustness.

1. Introduction

Given adequate data and computing power, deep neural networks (DNNs) have been demonstrated their potential to surpass human-level capability on image classification [10,34,35,38]. However, as is well-realized, DNNs still have plenty of flaws, one of which is their excessive sensitivity to adversarial examples [2,6,9,20,33]. In general, a primary branch of adversarial attacks is white-box attacks and black-box attacks. In white-box setting, all the information of the target model is known while there are no or limited knowledge in black-box setting. In many real world condition, black-box attacks are more challenging.

☆ The work is supported by the National Key Research and Development Program of China (2017YFB0202201, 2018YFB0203804, 2018YFB1701400), the National Natural Science Foundation of China (Grant Nos. 61873090, L1824034, L1924056), China Knowledge Centre for Engineering Sciences and Technology Project (CKCEST-2020–2-5, CKCEST-2019–2-13).

* Corresponding author at: College of Computer Science and Electronic Engineering, Hunan University, Changsha, China.
E-mail addresses: zhangchen2@hnu.edu.cn (C. Zhang), ztang@hnu.edu.cn (Z. Tang), yofei@hnu.edu.cn (Y. Zuo), lkl@hnu.edu.cn (K. Li), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.ins.2020.10.044
0020-0255/

Numerous black-box attacks have been proposed to fool the target model [3,6,22]. A common idea is to use a surrogate model instead of the target model to craft adversarial examples, which is termed as transfer-based attack. Su et al. have shown that adversarial examples transfer between models through sufficient experiments [31]. And Liu et al. have proposed transfer-based attack for real-world image classification system [18].

Nowadays, scholars have designed defense techniques against adversarial attacks to alleviate this security issue [19,24,28]. Among these, adversarial training is the most extensively investigated approach to increase the robustness of DNNs [13,36]. Meanwhile, generative models are proposed to detect adversarial examples [29] and improve robustness such as denoise auto-encoder [14].

However, there have been few recent studies on the robustness of generative classification models to adversarial examples, where such a classifier clearly created the conditional distribution of the input for given targets [16]. The classifier need to use Bayes' rule to transform a generated model into a discriminator, which makes predictions for a given input by comparing the probabilities of the labels. This is strongly linked to the "distance" of the input to the data manifold combined with a category. Therefore, generative classifiers ought to robust on many adversarial examples lately proposed if the "off-manifold" conjecture applies for many practical applications [16].

Inspired by recent researches [16,27], we propose a high robust classifier against transfer-based attacks under the framework of variational Auto-Encoder (VAE) which is one of the most prevalent generative models combined with deep learning. VAEs are successfully used to a large mount of tasks such as image generation [25], text generation [39], speech production [32], recommender system [15], and semi-supervised classification [42]. The encoder-decoder architecture of the VAEs allows it to learn the latent representation of high dimensional data inputs in the contiguous space, which makes the process of sampling from this latent space very straightforward.

Specifically, our model simulates the data generative process with several multivariate Gaussian distributions and DNNs. We assume that the latent embedding which are generated by an encoder (a DNN) of each category corresponds to a multivariate Gaussian distribution. Then a decoder (another DNN) is proposed to decode the latent representation into an observed variable. At last we can predict data's labels by chosing the largest posterior probability of all classes using Bayes' rule. Inference in our model is done in a variational way so we can utilize the Stochastic Gradient Variational Bayes (SGVB) estimator and reparamerization trick to optimize the evidence lower bound (ELBO). We quantitatively analyze the resistance of our model to transfer attacks, and experiments with quantitative comparisons show that our method reaches state-of-the-art (on MNIST and Fashion-MNIST datasets among all models, on CIFAR-10 dataset among generative classification models) with significantly better defense against transfer-based attacks. Our major contributions can be summarized as follows:

- We can get the ELBO of VAEs through calculating the mismatch between the true joint distribution and the variational joint distribution, which is very helpful for us to derive our model.
- We propose an image classification approach within the framework of VAE, which considering a Gaussian mixture prior on the latent variable.
- We quantitatively analyze the resistance of our model to transfer-based adversarial examples. Experimental results prove that our model reaches state-of-the-art with significantly better robustness.

This paper is organized as follows. The background of VAEs and adversarial attack and defense methods are given in Section 2. Our method is discussed in Section 3, where it mainly discusses the mechanism and robustness of our model. Experiments are implemented to illustrate the effectiveness of the proposed method in Section 4. The conclusion is given in Section 5.

2. Background

In this section we provide background on VAEs and some effective adversarial attack and defense methods on DNNs.

2.1. Variational Autoencoder

A VAE is composed of two networks that encode an input $\boldsymbol{x}$ to a latent variable $\boldsymbol{z}$ and decode the latent embedding which sampled by a posterior distribution back to the inputs space, respectively:

$$\boldsymbol{z} \sim \text{encoder}(\boldsymbol{x}; \Phi) = p(\boldsymbol{z}|\boldsymbol{x}), \tag{1}$$

$$\tilde{\boldsymbol{x}} \sim \text{decoder}(\boldsymbol{z}; \Theta) = q(\boldsymbol{x}|\boldsymbol{z}). \tag{2}$$

The VAEs regularize the *encoder* by applying a prior on the latent distribution $\tilde{p}(\boldsymbol{z})$ which generally chooses $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. VAEs estimate the log-likelihood $\log p(\boldsymbol{x})$ by learning a probabilistic generative model $p(\boldsymbol{x}|\boldsymbol{z})$ with latent variables $\boldsymbol{z}$:

$$\log p(\boldsymbol{x}) \geqslant \mathcal{L} = \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z}|\boldsymbol{x})}[\log q(\boldsymbol{x}|\boldsymbol{z})] - \text{KL}(p(\boldsymbol{z}|\boldsymbol{x})||\tilde{p}(\boldsymbol{z})), \tag{3}$$

where KL is the Kullback–Leibler divergence, and $\mathcal{L}$ is the so-called ELBO on the log-likelihood which consists of two parts: a reconstruction error and the mismatch between the true posterior and the variational.

2.2. Adversarial examples

Adversarial examples which were first proposed by Szegedy [33] are commonly found in DNNs. The pixel values of the input image change slightly, and the modified new image can still be distinguished by the human eyes, but the neural network model will misclassify it with high confidence. The changed image that leads to the misclassification is an adversarial example.

Liu et al. defined transferability that is the percentage of the adversarial examples generated from *source model* to misclassify *target model* [18]. Transfer attacks are based on the transferability of DNNs.

For mathematical definition, we want to find a minimum perturbation term $\boldsymbol{\delta}$ which is added to the original sample $\boldsymbol{x}$ to get an adversarial example $\boldsymbol{x}^*$. In general, the method of seeking adversarial example can be turned into an optimization problem:

$$\begin{cases} \boldsymbol{\delta} = \min \|\boldsymbol{x} - \boldsymbol{x}^*\|_p \\ \text{s.t.} \quad f(\boldsymbol{x}^*) \neq f(\boldsymbol{x}), \end{cases} \tag{4}$$

where $f(\cdot)$ is a classification result of deep neural network model and $\|\cdot\|_p$ is the ℓ_p norm of a vector.

If the target model structure and parameters are known, we can get the adversarial examples by gradient-based method such as the fast gradient sign method (FGSM) [9] when there is ℓ_∞ norm:

$$\boldsymbol{\delta} = \epsilon \operatorname{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y)), \tag{5}$$

where $\|\boldsymbol{x}^* - \boldsymbol{x}\|_\infty \leqslant \epsilon$, and $J(\boldsymbol{\theta}, \boldsymbol{x}, y)$ is the loss function of the target model. An adversarial example can be generated by FGSM through performing gradient calculations once, which is very fast. But the success rate of FGSM sometimes is not very high. To enhance FGSM's performance, Kurakin et al. proposed iterative FGSM (I-FGSM) [13] that iteratively calculated FGSM with a finer direction and could be regarded as a projected gradient descent (PGD) method inside an ℓ_∞ ball [4].

To improve the transferability, Wu et al. presented Variance-Reduced Iterative FGSM (Vr-IGSM) which utilized an averaged gradient of original input with Gaussian noises to eliminate local fluctuation in surrogate model [41]:

$$\begin{aligned} G_t &= \tfrac{1}{m} \sum_{i=1}^{m} \nabla J(\boldsymbol{x}'_t + \xi_i), \qquad \xi_i \in \mathcal{N}(0, \sigma^2 \boldsymbol{I}) \\ \boldsymbol{x}'_{t+1} &= \operatorname{Clip}\big(\boldsymbol{x}'_t + \epsilon \cdot \operatorname{sign}(G_t)\big). \end{aligned} \tag{6}$$

Another strong attack MI-FGSM utilized the momentum method which accumulates a velocity vector in the gradient direction of the loss function across iterations for crafting adversarial examples [6]. This attack can fool DNN models effectively, even in black-box manner. It outperforms one-step gradient-based and vanilla iterative approaches.

$$\begin{aligned} \boldsymbol{g}_{t+1} &= \mu \cdot \boldsymbol{g}_t + \tfrac{\nabla_{\boldsymbol{x}} J(\boldsymbol{x}'_t, y)}{\|\nabla_{\boldsymbol{x}} J(\boldsymbol{x}'_t, y)\|_1} \\ \boldsymbol{x}'_{t+1} &= \operatorname{Clip}\big(\boldsymbol{x}'_t + \epsilon \cdot \operatorname{sign}(\boldsymbol{g}_{t+1})\big). \end{aligned} \tag{7}$$

When there is ℓ_2 norm, we can get adversarial examples by deepfool method [20]. It generates an adversarial example $\boldsymbol{x}^*$ by finding the distance between the initial sample $\boldsymbol{x}$ and the demarcated hyperplane by the steepest descent method:

$$\begin{aligned} \boldsymbol{\delta} &= l(\boldsymbol{x})(\nabla f_l(\boldsymbol{x}) - \nabla f_k(\boldsymbol{x})) \\ l(\boldsymbol{x}) &= \operatorname*{argmin}_{k' \neq k} \tfrac{|f_{k'}(\boldsymbol{x}) - f_k(\boldsymbol{x})|}{\|\nabla f_{k'}(\boldsymbol{x}) - \nabla f_k(\boldsymbol{x})\|_2}, \end{aligned} \tag{8}$$

where $f_k(\boldsymbol{x})$ is the output of $f(\boldsymbol{x})$ before softmax layer that corresponds to the k^{th} class. The DeepFool method can simply be adapted to find minimal adversarial perturbations for any ℓ_p norm ($p \in [1, +\infty]$).

When there is ℓ_0 norm, we can get adversarial examples by JSMA (Jacobian-based Saliency Map Attack) method [23]. The method is a greedy algorithm that increases the probability of target class for each iteration by picking pixels to modify each time. The attack utilizes the gradient $\nabla Z(\boldsymbol{x})_l$to get a *saliency map*, which calculates the effect of each pixel of classification results, where $Z(\boldsymbol{x})$ is the output of all network layers except the softmax and l is the target class. For the saliency map, this method repeatedly selects the most significant pixel and modifies it to raise the likelihood of the class l until either the modified pixels exceed a set threshold, which leads the attack unsuccessful, or changes the classification effectively. Particularly, the method defines the saliency map with pair of pixels p, q. Definition α_{pq} represents the impact of changes in both pixels p and q on target classification, and β_{pq} indicates how much all other outputs p and q will change:

$$\alpha_{pq} = \sum_{i \in \{p,q\}} \frac{\partial Z(\boldsymbol{x})_l}{\partial \boldsymbol{x}_i}, \tag{9}$$

$$\beta_{pq} = \left(\sum_{i \in \{p,q\}} \sum_{j} \frac{\partial Z(\boldsymbol{x})_j}{\partial \boldsymbol{x}_i} \right) - \alpha_{pq}. \tag{10}$$

Then the approach applies the following equation to get the pixels:

$$(p^*, q^*) = \arg\max_{p,q}(-\alpha_{pq} \cdot \beta_{pq})(\alpha_{pq} > 0) \cdot (\beta_{pq} < 0), \tag{11}$$

where $\alpha_{pq} > 0$means the target classification is more possible, $\beta_{pq} < 0$means the other categories are less possible, and $-\alpha_{pq} \cdot \beta_{pq}$is largest.

C&W attacks [2] formulate the generating adversarial examples problem by adding a cost function which evaluates the difference between the prediction of input and the target label to the original problem:

$$\text{minimize}\|\boldsymbol{x}^* - \boldsymbol{x}\|_p^2 + c \cdot f(\boldsymbol{x}^*) \tag{12}$$

with $f$defined as

$$f(\boldsymbol{x}^*) = \max\left(\max\{Z(\boldsymbol{x}^*)_i : i \neq l\} - Z(\boldsymbol{x}^*)_l, -\kappa\right), \tag{13}$$

where κ is a parameter that encourages the solver to detect an adversarial example $\boldsymbol{x}^*$ with a high possibility as class l that target model outputs. We let $\kappa = 0$ for our experiments as the authors done. Because ℓ_0 is not differentiable, the ℓ_0 method is conducted by iteratively removing pixels which are trivial for generating adversarial examples by gradient of ℓ_2 distance. C&W attacks are among the most effective attacks that find adversarial examples with small ℓ_2, ℓ_∞, and ℓ_0 disturbances. So these attacks can often be used to assess the effectiveness of potential defenses. The two authors also encouraged those who propose defenses implement both assessment methods: (1) Take advantage of a powerful attack; (2) Demonstrate that transferability fails.

2.3. Defense methods

For the purpose of defending against adversarial examples, many scholars have proposed a series of defense methods. Defensive distillation [24] and gradient masking based methods [5] attempted to optimize the gradient of models against adversarial attacks. However, they are vulnerable to approximated gradient attacks [1,17]. Recently, adversarial training has been proven to be the most extensively investigated way to improve the robustness of deep learning models [36,37,40]. It can be seen as the following robust optimization problem:

$$\begin{aligned} &\min_\theta \sum_i \max_{\delta \in \Delta} \ell(f_\theta(x_i + \delta), y_i), \\ &\Delta = \{\delta : ||\delta||_\infty \leqslant \epsilon\}. \end{aligned} \tag{14}$$

In addition, detecting and purifying adversarial examples methods are effective from another way to protect DNN models [7,11,30]. Detecting adversarial images or decontaminating the imperceptible perturbations before importing data into DNNs can also increase the accuracy of deep classifiers. Because the strong similarity and correlation between adjacent pixels in the local structure of figures, image compression can reduce the redundant information of the image while retaining the significance. Thus, in [11], the authors designed ComDefend, which utilized image compression to eliminate or break the structure of disturbances.

3. Variational deep embedding for classification

In this section, we represent our classifier, a probabilistic robust classification model based on the framework of VAEs.

First of all, we find a novel way to get the ELBO of VAEs, which is helpful for us to derive our model. We can calculate the KL divergence of two joint probabilities $p(\boldsymbol{x}, \boldsymbol{z})$ and $q(\boldsymbol{x}, \boldsymbol{z})$ by the following equation:

$$\begin{aligned} &\text{KL}(p(\boldsymbol{x}, \boldsymbol{z})||q(\boldsymbol{x}, \boldsymbol{z})) \\ &= \mathbb{E}_{\boldsymbol{x}\sim\tilde{p}(\boldsymbol{x})}\left[\int_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{x}) \log \frac{\tilde{p}(\boldsymbol{x})p(\boldsymbol{z}|\boldsymbol{x})}{q(\boldsymbol{x}, \boldsymbol{z})} d\boldsymbol{z}\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim\tilde{p}(\boldsymbol{x})}\left[\log \tilde{p}(\boldsymbol{x}) \int_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{x}) d\boldsymbol{z} + \int_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p(\boldsymbol{z}|\boldsymbol{x})}{q(\boldsymbol{x}, \boldsymbol{z})} d\boldsymbol{z}\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim\tilde{p}(\boldsymbol{x})}[\log \tilde{p}(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim\tilde{p}(\boldsymbol{x})}\left[\int_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p(\boldsymbol{z}|\boldsymbol{x})}{q(\boldsymbol{x}, \boldsymbol{z})} d\boldsymbol{z}\right] \\ &= C + \mathbb{E}_{\boldsymbol{x}\sim\tilde{p}(\boldsymbol{x})}\left[\int_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p(\boldsymbol{z}|\boldsymbol{x})}{q(\boldsymbol{z})q(\boldsymbol{x}|\boldsymbol{z})} d\boldsymbol{z}\right] \\ &= C + \mathbb{E}_{\boldsymbol{x}\sim\tilde{p}(\boldsymbol{x})}\left[\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z}|\boldsymbol{x})}[-\log q(\boldsymbol{x}|\boldsymbol{z})] + \text{KL}(p(\boldsymbol{z}|\boldsymbol{x})||\tilde{p}(\boldsymbol{z}))\right] \\ &= -\mathcal{L}, \end{aligned} \tag{15}$$

where we have omitted the constant term, and $\mathbb{E}_{\boldsymbol{x}\sim\tilde{p}(\boldsymbol{x})}[\cdot]$ can also be omitted because $\tilde{p}(\boldsymbol{x})$ is the natural distribution of $\boldsymbol{x}$. So we prove that maximizing ELBO of VAEs is equivalent to minimizing $\text{KL}(p(\boldsymbol{x}, \boldsymbol{z})||q(\boldsymbol{x}, \boldsymbol{z}))$.

3.1. The generative process

Since we solve the classification problem by a type of generative model, we first represent the generative procedure of our method. The framework of our model is shown in Fig. 1. The main idea is considering a Gaussian mixture prior on the latent variable while training the VAE. Then at the inference time, we utilize Bayes' rule to get the classification.

Specifically, assume there are N categories, the model generates an observed input $\boldsymbol{x} \in \mathbb{R}^d$ by the process:

1. Pick a category $y \sim p(Y)$
2. Pick a latent vector $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}_y, \boldsymbol{\sigma}_y^2 \boldsymbol{I})$
3. Pick an input $\boldsymbol{x}$:
   a If $\boldsymbol{x}$ is binary
      i Compute the expectation vector $\boldsymbol{\mu}_x$

$$\boldsymbol{\mu}_x = \text{decoder}(\boldsymbol{z}; \boldsymbol{\theta}) \tag{16}$$

      ii Pick an input $\boldsymbol{x} \sim \text{Bernoulli}(\boldsymbol{\mu}_x)$
   b If $\boldsymbol{x}$ is real valued
      i Computer $\boldsymbol{\mu}_x$ and $\boldsymbol{\sigma}_x^2$

$$[\boldsymbol{\mu}_x; \log \boldsymbol{\sigma}_x^2] = \text{decoder}(\boldsymbol{z}; \boldsymbol{\Theta}) \tag{17}$$

      ii Pick an input $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\sigma}_x^2 \boldsymbol{I})$

where we assume $p(Y)$ is a discrete uniform distribution; $\boldsymbol{\mu}_y$ and $\boldsymbol{\sigma}_y^2$ are the mean and the variance of the Gaussian distribution corresponding to the category y which we should to get by training data; $\boldsymbol{I}$ is an identity matrix; decoder$(\boldsymbol{z}; \boldsymbol{\theta})$, parametrized by $\boldsymbol{\theta}$, is a DNN whose input is $\boldsymbol{z}$; Bernoulli$(\boldsymbol{\mu}_x)$ and $\mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\sigma}_x^2 \boldsymbol{I})$are multivariate Bernoulli distribution and Gaussian distribution which are parametrized by $\boldsymbol{\mu}_x$ and $\boldsymbol{\mu}_x, \boldsymbol{\sigma}_x^2$ respectively. On the basis of above generation procedure, the joint probability $q(\boldsymbol{x}, \boldsymbol{z}, y)$ can be factored as:

$$q(\boldsymbol{x}, \boldsymbol{z}, y) = q(\boldsymbol{x}|\boldsymbol{z}) q(\boldsymbol{z}|y) q(y). \tag{18}$$

We assume that $\boldsymbol{x}$ and y are independent conditioned on $\boldsymbol{z}$ and the label distribution $q(y)$ can be evaluated from the training data. But in general, in order to balance each category, we also take a uniform distribution to $q(y)$. So the probabilities are defined as:

$$q(y) = p(Y, N) = \frac{1}{N}, Y = 1, \cdots, N, \tag{19}$$

$$q(\boldsymbol{z}|y) = \mathcal{N}(\boldsymbol{\mu}_y, \boldsymbol{\sigma}_y^2 \boldsymbol{I}), \tag{20}$$

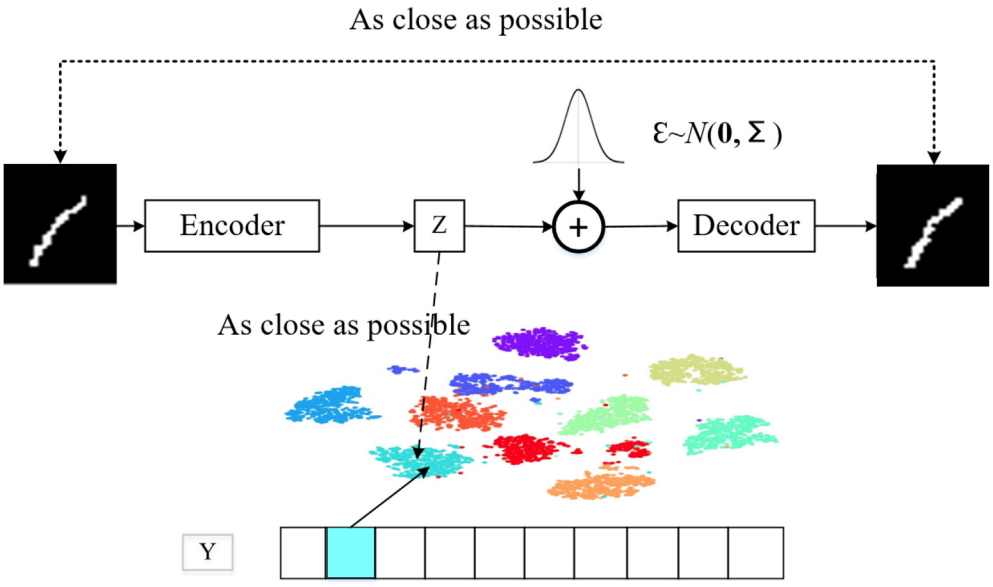

Fig. 1. The framework of our model.

$$q(\boldsymbol{x}|\boldsymbol{z}) = \text{Bernoulli}(\boldsymbol{\mu}_x) \quad \text{or} \quad \mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\sigma}_x^2 \boldsymbol{I}), \tag{21}$$

3.2. Variational Lower Bound

A classifier of our approach is turned to maximize the likelihood of the given data. We could find ELBO by variational inference theory as same as VAE does. As we have proved in the beginning of this section, maximize the ELBO is equivalent to minimize the KL-divergence of the two distributions. So given the generation process in Section 3.1 and as same as Eq. (18), the joint probability $p(\boldsymbol{x}, \boldsymbol{z}, y)$ can be factored as:

$$p(\boldsymbol{x}, \boldsymbol{z}, y) = p(y|\boldsymbol{z})p(\boldsymbol{z}|\boldsymbol{x})p(\boldsymbol{x}). \tag{22}$$

The KL-divergence of the two joint probability distributions can be written as:

$$\begin{aligned} D_{KL} &= \text{KL}(p(\boldsymbol{x}, \boldsymbol{z}, y)||q(\boldsymbol{x}, \boldsymbol{z}, y)) \\ &= \sum_y \int_{\boldsymbol{x}} \int_{\boldsymbol{z}} p(\boldsymbol{x}, \boldsymbol{z}, y) \log \tfrac{p(\boldsymbol{x}, \boldsymbol{z}, y)}{q(\boldsymbol{x}, \boldsymbol{z}, y)} d\boldsymbol{z} d\boldsymbol{x} \\ &= \sum_y \int_{\boldsymbol{x}} \int_{\boldsymbol{z}} p(y|\boldsymbol{z})p(\boldsymbol{z}|\boldsymbol{x})\tilde{p}(\boldsymbol{x}) \log \tfrac{p(y|\boldsymbol{z})p(\boldsymbol{z}|\boldsymbol{x})\tilde{p}(\boldsymbol{x})}{q(y)q(\boldsymbol{z}|y)q(\boldsymbol{x}|\boldsymbol{z})} d\boldsymbol{z} d\boldsymbol{x}, \end{aligned} \tag{23}$$

where $q(\boldsymbol{x}, \boldsymbol{z}, y)$ is the variational joint probability that approximates the true joint probability $p(\boldsymbol{x}, \boldsymbol{z}, y)$. And $\tilde{p}(\boldsymbol{x})$ is the natural distribution of given data $\boldsymbol{x} = \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(m)}$ that we want to get, but it's difficult to get generally. In our approach, we assume $q(y|\boldsymbol{z})$ to be a discrete uniform distribution, the same distribution as $q(y)$. Then the KL-divergence in Eq. (23) can be rewritten as:

$$\begin{aligned} &\text{KL}(p(\boldsymbol{x}, \boldsymbol{z}, y)||q(\boldsymbol{x}, \boldsymbol{z}, y)) \\ &= \mathbb{E}_{\boldsymbol{x}\sim\tilde{p}(\boldsymbol{x})}\left[\sum_y \int_{\boldsymbol{z}} p(y|\boldsymbol{z})p(\boldsymbol{z}|\boldsymbol{x}) \log \tfrac{p(y|\boldsymbol{z})p(\boldsymbol{z}|\boldsymbol{x})\tilde{p}(\boldsymbol{x})}{q(y)q(\boldsymbol{z}|y)q(\boldsymbol{x}|\boldsymbol{z})} d\boldsymbol{z}\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim\tilde{p}(\boldsymbol{x})}\left[\int_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{x}) \log \tfrac{p(y|\boldsymbol{z})p(\boldsymbol{z}|\boldsymbol{x})\tilde{p}(\boldsymbol{x})}{q(y)q(\boldsymbol{z}|y)q(\boldsymbol{x}|\boldsymbol{z})} d\boldsymbol{z}\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim\tilde{p}(\boldsymbol{x})}\left[\int_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{x})\left(\log \tilde{p}(\boldsymbol{x}) - \log q(\boldsymbol{x}|\boldsymbol{z}) + \log \tfrac{p(\boldsymbol{z}|\boldsymbol{x})}{q(\boldsymbol{z}|y)} + \log \tfrac{p(y|\boldsymbol{z})}{q(y)}\right) d\boldsymbol{z}\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim\tilde{p}(\boldsymbol{x})}[\log \tilde{p}(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim\tilde{p}(\boldsymbol{x})}\left[\int_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{x})\left(-\log q(\boldsymbol{x}|\boldsymbol{z}) + \log \tfrac{p(\boldsymbol{z}|\boldsymbol{x})}{q(\boldsymbol{z}|y)} + \log \tfrac{p(y|\boldsymbol{z})}{q(y)}\right) d\boldsymbol{z}\right] \\ &= C + \mathbb{E}_{\boldsymbol{z}\sim\boldsymbol{p}(\boldsymbol{z}|\boldsymbol{x})}[-\log q(\boldsymbol{x}|\boldsymbol{z}) + \text{KL}(p(\boldsymbol{z}|\boldsymbol{x})||q(\boldsymbol{z}|y))]. \end{aligned} \tag{24}$$

In our method, similar to original VAE, we use a DNN encoder to model $p(\boldsymbol{z}|\boldsymbol{x})$:

$$[\tilde{\boldsymbol{\mu}}; \log \tilde{\boldsymbol{\sigma}}^2] = \text{encoder}(\boldsymbol{x}; \Phi), \tag{25}$$

$$p(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\sigma}}^2 \boldsymbol{I}), \tag{26}$$

where Φ is the parameter of network encoder.

By substituting the terms in Eq. (24) with Eqs. (19), (20), (21), and (26), using the SGVB estimator and the reparameterization trick, the $\text{KL}(p(\boldsymbol{x}, \boldsymbol{z}, y)||q(\boldsymbol{x}, \boldsymbol{z}, y))$ can be rewritten as:

(a) If $\boldsymbol{x}$ is binary

$$\begin{aligned} &\text{KL}(p(\boldsymbol{x}, \boldsymbol{z}, y)||q(\boldsymbol{x}, \boldsymbol{z}, y)) \\ &= -\tfrac{1}{2K}\sum_{k=1}^{K}\sum_{j=1}^{J}\left(x_j \log \boldsymbol{\mu}_{\boldsymbol{x}}^{(k)}|_j + (1 - x_j) \log\left(1 - \boldsymbol{\mu}_{\boldsymbol{x}}^{(k)}|_j\right)\right) + \tfrac{1}{2}\sum_{l=1}^{L}\left(\log \tfrac{\boldsymbol{\sigma}_{y|l}^2}{\tilde{\boldsymbol{\sigma}}_l^2} + \tfrac{1}{\boldsymbol{\sigma}_{y|l}^2}\left(\tilde{\boldsymbol{\mu}}_l - \boldsymbol{\mu}_y|_l\right)^2 + \tfrac{\tilde{\boldsymbol{\sigma}}_l^2}{\boldsymbol{\sigma}_{y|l}^2} - 1\right). \end{aligned} \tag{27}$$

(b) If $\boldsymbol{x}$ is real valued

$$\begin{aligned} &\text{KL}(p(\boldsymbol{z}|\boldsymbol{x})||q(\boldsymbol{z}|y)) \\ &= \int_{-\infty}^{+\infty} \tfrac{1}{\sqrt{2\pi\tilde{\boldsymbol{\sigma}}}} \exp\left(-\tfrac{(\boldsymbol{x}-\tilde{\boldsymbol{\mu}})^2}{2\tilde{\boldsymbol{\sigma}}^2}\right) \log\left[\tfrac{1}{\sqrt{2\pi}\tilde{\boldsymbol{\sigma}}} \bullet \exp\left(-\tfrac{(\boldsymbol{x}-\tilde{\boldsymbol{\mu}})^2}{2\tilde{\boldsymbol{\sigma}}^2}\right) / \tfrac{1}{\sqrt{2\pi\boldsymbol{\sigma}_y}} \exp\left(-\tfrac{(\boldsymbol{x}-\boldsymbol{\mu}_y)^2}{2\boldsymbol{\sigma}_y^2}\right)\right] d\boldsymbol{x} \\ &= \int_{-\infty}^{+\infty} \tfrac{1}{\sqrt{2\pi\tilde{\boldsymbol{\sigma}}}} \exp\left(-\tfrac{(\boldsymbol{x}-\tilde{\boldsymbol{\mu}})^2}{2\tilde{\boldsymbol{\sigma}}^2}\right) \bullet \tfrac{1}{2}\left[\log \tfrac{\boldsymbol{\sigma}_y^2}{\tilde{\boldsymbol{\sigma}}^2} + \tfrac{1}{\boldsymbol{\sigma}_y^2}\left((\boldsymbol{x} - \boldsymbol{\mu}_y)^2 - \tfrac{\boldsymbol{\sigma}_y^2(\boldsymbol{x}-\tilde{\boldsymbol{\mu}})^2}{\tilde{\boldsymbol{\sigma}}^2}\right)\right] d\boldsymbol{x} \\ &= \tfrac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\sigma}}^2)}\left[\log \tfrac{\boldsymbol{\sigma}_y^2}{\tilde{\boldsymbol{\sigma}}^2} + \tfrac{1}{\boldsymbol{\sigma}_y^2}\left(\boldsymbol{x}^2 - 2\boldsymbol{x}\boldsymbol{\mu}_y + \boldsymbol{\mu}_y^2 - \tfrac{\boldsymbol{\sigma}_y^2(\boldsymbol{x}-\tilde{\boldsymbol{\mu}})^2}{\tilde{\boldsymbol{\sigma}}^2}\right)\right] \\ &= \tfrac{1}{2}\sum_{l=1}^{L}\left(\log \tfrac{\boldsymbol{\sigma}_{y|l}^2}{\tilde{\boldsymbol{\sigma}}_l^2} + \tfrac{1}{\boldsymbol{\sigma}_{y|l}^2}\left(\tilde{\boldsymbol{\mu}}_l - \boldsymbol{\mu}_y|_l\right)^2 + \tfrac{\tilde{\boldsymbol{\sigma}}_l^2}{\boldsymbol{\sigma}_{y|l}^2} - 1\right). \end{aligned} \tag{28}$$

$$\begin{aligned}&\mathrm{KL}(p(\boldsymbol{x},\boldsymbol{z},y)||q(\boldsymbol{x},\boldsymbol{z},y))\\&=\frac{1}{2K}\sum_{k=1}^{K}\sum_{j=1}^{J}\left(\frac{1}{\sigma_x^2|_j}\left(x_j-\mu_{\boldsymbol{x}}^{(k)}|_j\right)^2+\log\sigma_x^2|_j\right)+\frac{1}{2}\sum_{l=1}^{L}\left(\log\frac{\sigma_{y}^2|_l}{\tilde{\sigma}_l^2}+\frac{1}{\sigma_y^2|_l}\left(\tilde{\mu}_l-\mu_y|_l\right)^2+\frac{\tilde{\sigma}_l^2}{\sigma_y^2|_l}-1\right),\end{aligned}\tag{29}$$

where K is the number of Monte Carlo samples in the SGVB estimator, J is the dimensionality of $\boldsymbol{x}$ and $\boldsymbol{\mu}_{\boldsymbol{x}}^{(k)}, x_j$ is the j^{th} element of $\boldsymbol{x}$, $*|_j$ denotes the j^{th} element of $*$, L is the dimensionality of $\boldsymbol{\mu}_y, \boldsymbol{\sigma}_y^2, \tilde{\boldsymbol{\sigma}}$, and $\tilde{\boldsymbol{\sigma}}^2$. In Eq. (29), we also have omitted the constant term, and $\boldsymbol{\sigma}_x^2, \boldsymbol{\sigma}_y^2$ can be treated as hyperparameters or derived from neural networks training.

In Eq. (29), we compute $\boldsymbol{\mu}_{\boldsymbol{x}}^{(k)}$ as:

$$\boldsymbol{\mu}_{\boldsymbol{x}}^{(k)}=\mathrm{decoder}(\boldsymbol{z}^{(k)};\boldsymbol{\Theta}),\tag{30}$$

where $\boldsymbol{z}^{(k)}$ is the k^{th} sample from $p(\boldsymbol{z}|\boldsymbol{x})$ by Eq. (26) to produce the Monte Carlo samples. According to the reparameterization trick by [12], $\boldsymbol{z}^{(k)}$ is obtained by:

$$\boldsymbol{z}^{(k)}=\tilde{\boldsymbol{\mu}}+\tilde{\boldsymbol{\sigma}}\circ\boldsymbol{\epsilon}^{(k)},\tag{31}$$

where $\boldsymbol{\epsilon}^{(k)}\sim\mathcal{N}(\boldsymbol{0},\boldsymbol{I}), \circ$ is the operator of element-wise multiplication, and $\boldsymbol{\mu}_y, \boldsymbol{\sigma}_y$ are derived by Eq. (20). The summary of our approach training is shown in Algorithm 1.

Algorithm 1 Training our model. In experiments, we set $M = 256$ and $L = 1$.

1: $\boldsymbol{\mu}_y, \boldsymbol{\Theta}, \boldsymbol{\Phi} \leftarrow$ Initialize the parameters
2: **Repeat**
3: $\boldsymbol{x}^M \leftarrow$Random minibatch of M inputs (drawn from full dataset)
4: $\epsilon \leftarrow$Random samples from noise distribution $\mathcal{N}(\boldsymbol{0},\boldsymbol{I})$
5: $g \leftarrow \nabla_{\boldsymbol{\mu}_y,\boldsymbol{\Theta},\boldsymbol{\Phi}}\mathrm{KL}(p(\boldsymbol{x},\boldsymbol{z},y)||q(\boldsymbol{x},\boldsymbol{z},y))$ (Gradients of minibatch estimator)
6: $\boldsymbol{\mu}_y, \boldsymbol{\Theta}, \boldsymbol{\Phi} \leftarrow$ Update parameters using gradients g (e.g. Adam or SGD)
7: **Until** convergence of parameters $(\boldsymbol{\mu}_y, \boldsymbol{\Theta}, \boldsymbol{\Phi})$
8: **Return** result

3.3. Understanding the KL Divergence of Our Approach

In this section, some intuitions of the $\mathrm{KL}(p(\boldsymbol{x},\boldsymbol{z},y)||q(\boldsymbol{x},\boldsymbol{z},y))$ of our method are provided. More specifically, Eq. (24) can be simplified as follows:

$$\mathrm{KL}(p(\boldsymbol{x},\boldsymbol{z},y)||q(\boldsymbol{x},\boldsymbol{z},y))=\mathbb{E}_{\boldsymbol{z}\sim\boldsymbol{p}(\boldsymbol{z}|\boldsymbol{x})}[-\log q(\boldsymbol{x}|\boldsymbol{z})+\mathrm{KL}(p(\boldsymbol{z}|\boldsymbol{x})||q(\boldsymbol{z}|y))].\tag{32}$$

The left hand side in the right term of Eq. (32) is the *reconstruction* term, which encourages our model to explain the dataset well while the right is the KL divergence from the prior $q(\boldsymbol{z}|y)$ to the variational posterior $p(\boldsymbol{z}|\boldsymbol{x})$, which regularizes the latent representation $\boldsymbol{z}$ to lie on a Mixture-of-Gaussian manifold.

3.4. Classification Process

Once the training is done by minimize the KL-divergence w.r.t the parameters of $\{\boldsymbol{\mu}_y, \boldsymbol{\sigma}_y, \boldsymbol{\Theta}, \boldsymbol{\Phi}\}, y \in 1, 2, \cdots, N$, a latent embedding $\boldsymbol{z}$ can be extracted for each observed input $\boldsymbol{x}$ by Eqs. (25) and (26). Since the posterior $p(y|\boldsymbol{x})$ is difficult to learn directly, we use the following equation to learn generative distribution $p(\boldsymbol{x}|y)$ and classify new samples by Bayes' rule:

$$p(y|\boldsymbol{x})\quad=\frac{q(\boldsymbol{x}|y)q(y)}{\tilde{p}(\boldsymbol{x})}=q(\boldsymbol{x}|\boldsymbol{z})q(\boldsymbol{z}|y)\frac{q(y)}{\tilde{p}(\boldsymbol{x})},\tag{33}$$

where $\tilde{p}(\boldsymbol{x})$ is the natural distribution of $\boldsymbol{x}$. And we can get $q(y)$ by Eq. (19) or from the dataset, get $q(\boldsymbol{z}|y)$ by Eq. (20), and get $p(\boldsymbol{z}|\boldsymbol{x})$ by Eq. (26).

Thus, for each class y we train the VAE to learn the class-conditional distribution $q(\boldsymbol{x}|y)$. This enables us to evaluate ELBO on the log-likelihood $\log p(\boldsymbol{x})$ of input $\boldsymbol{x}$ for each class y.

$$\log p(\boldsymbol{x})\geqslant\mathcal{L}=-\mathrm{KL}(p(\boldsymbol{x},\boldsymbol{z},y)||q(\boldsymbol{x},\boldsymbol{z},y)),\tag{34}$$

We use variational inference during training and perform "exact" inference over $q(\boldsymbol{x}|\boldsymbol{z})$ during evalution. This inference is implemented to find the optimal $\boldsymbol{z}$ which maximizes the ELBO on the log-likelihood for each class with input $\boldsymbol{x}$:

$$\begin{aligned}\mathcal{L}_y(\boldsymbol{x}) &= \max_{\boldsymbol{z}}[-\mathrm{KL}(p(\boldsymbol{x},\boldsymbol{z},y)||q(\boldsymbol{x},\boldsymbol{z},y))]\\ &= \max_{\boldsymbol{z}}[\log q(\boldsymbol{x}|\boldsymbol{z}) - \mathrm{KL}(p(\boldsymbol{z}|\boldsymbol{x})||q(\boldsymbol{z}|y))]\\ &= \max_{\tilde{\boldsymbol{\mu}}}\Big[\log q(\boldsymbol{x}|\boldsymbol{z}) - \mathrm{KL}\Big(\mathcal{N}(\tilde{\boldsymbol{\mu}},\tilde{\boldsymbol{\sigma}}^2\boldsymbol{I}).||\mathcal{N}(\boldsymbol{\mu}_y,\boldsymbol{\sigma}_y^2\boldsymbol{I})\Big)\Big],\end{aligned} \tag{35}$$

where $\boldsymbol{z}$ in $\log q(\boldsymbol{x}|\boldsymbol{z})$ obeys a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_y,\boldsymbol{\sigma}_y^2\boldsymbol{I})$.

Finally, to implement the actual classification, all $\mathcal{L}_y(\boldsymbol{x})$ are scaled with a factor α, exponentiate, add an offset η and divide by the total evidence:

$$p(y|\boldsymbol{x}) = \big(e^{\alpha\mathcal{L}_y(\boldsymbol{x})}+\eta\big)/\sum_c\big(e^{\alpha\mathcal{L}_c(\boldsymbol{x})}+\eta\big). \tag{36}$$

The reason why we introduce η is derived from [27].

3.5. Classifier's robustness

The determination of our model relies on the likelihood of each category. For clean inputs, that is mainly determined by the posterior likelihood $q(\boldsymbol{x}|\boldsymbol{z})$ which depends on whether $\boldsymbol{x}$ is binary or real valued. We choose this posterior to be Gaussian normally. The class conditional likelihood can only change elegantly with changes in $\boldsymbol{x}$, so we can derive the lower bounds of our model robustness. As seen this, note that Eq. (35) can be rewritten as:

$$\mathcal{L}_y(\boldsymbol{x}) = \max_{\tilde{\boldsymbol{\mu}}}\left[-\mathrm{KL}\Big(\mathcal{N}(\tilde{\boldsymbol{\mu}},\tilde{\boldsymbol{\sigma}}^2\boldsymbol{I})||\mathcal{N}(\boldsymbol{\mu}_y,\boldsymbol{\sigma}_y^2\boldsymbol{I})\Big) - \frac{1}{2\boldsymbol{\sigma}_x^2}\|\mathrm{decoder}(\boldsymbol{z})-\boldsymbol{x}\|_2^2 + C\right],\ \boldsymbol{z}\sim\mathcal{N}(\boldsymbol{\mu}_y,\boldsymbol{\sigma}_y^2\boldsymbol{I}), \tag{37}$$

where we absorb the normalization constants of $q(\boldsymbol{x}|\boldsymbol{z})$ into C and decoder$(\boldsymbol{z})$ is the mean of $q(\boldsymbol{x}|\boldsymbol{z},y)$. Let y_T be the ground truth class and let $\tilde{\boldsymbol{\mu}}_{\boldsymbol{x}}^*$ be the optimal latent for the clean sample $\boldsymbol{x}$ for class y_T. A lower bound on $\mathcal{L}_y(\boldsymbol{x}+\boldsymbol{\delta})$ for a disturbance $\boldsymbol{\delta}$ with size $\epsilon = \|\boldsymbol{\delta}\|_2$ can be estimated:

$$\begin{aligned}\mathcal{L}_{y_T}(\boldsymbol{x}+\boldsymbol{\delta}) &= \max_{\tilde{\boldsymbol{\mu}}}\Big[-\mathrm{KL}\Big(\mathcal{N}(\tilde{\boldsymbol{\mu}},\tilde{\boldsymbol{\sigma}}^2\boldsymbol{I}).||\mathcal{N}(\boldsymbol{\mu}_y,\boldsymbol{\sigma}_y^2\boldsymbol{I})\Big) - \tfrac{1}{2\boldsymbol{\sigma}_x^2}\|\mathrm{decoder}(\boldsymbol{z})-\boldsymbol{x}-\boldsymbol{\delta}\|_2^2 + C\Big]\\ &\geqslant -\mathrm{KL}\Big(\mathcal{N}(\tilde{\boldsymbol{\mu}}^*,\tilde{\boldsymbol{\sigma}}^2\boldsymbol{I}).||\mathcal{N}(\boldsymbol{\mu}_{y_T},\boldsymbol{\sigma}_{y_T}^2\boldsymbol{I})\Big) - \tfrac{1}{2\boldsymbol{\sigma}_x^2}\|\mathrm{decoder}(\boldsymbol{z})-\boldsymbol{x}-\boldsymbol{\delta}\|_2^2 + C\\ &= \mathcal{L}_{y_T}(\boldsymbol{x}) + \tfrac{1}{\boldsymbol{\sigma}_x^2}\boldsymbol{\delta}^T(\mathrm{decoder}(\boldsymbol{z})-\boldsymbol{x}) - \tfrac{1}{2\boldsymbol{\sigma}_x^2}\epsilon^2 + C\\ &\geqslant \mathcal{L}_{y_T}(\boldsymbol{x}) - \tfrac{1}{\boldsymbol{\sigma}_x^2}\epsilon\|\mathrm{decoder}(\boldsymbol{z})-\boldsymbol{x}\|_2 - \tfrac{1}{2\boldsymbol{\sigma}_x^2}\epsilon^2 + C\\ &\boldsymbol{z}\sim\mathcal{N}(\boldsymbol{\mu}_{y_T},\boldsymbol{\sigma}_{y_T}^2\boldsymbol{I}).\end{aligned} \tag{38}$$

Similarly, an upper bound of $\mathcal{L}_y(\boldsymbol{x}+\boldsymbol{\delta})$ for all other class $y\neq y_T$ could be derived:

$$\begin{aligned}\mathcal{L}_y(\boldsymbol{x}+\boldsymbol{\delta}) &= \max_{\tilde{\boldsymbol{\mu}}}\Big[-\mathrm{KL}\Big(\mathcal{N}(\tilde{\boldsymbol{\mu}},\tilde{\boldsymbol{\sigma}}^2\boldsymbol{I}).||\mathcal{N}(\boldsymbol{\mu}_y,\boldsymbol{\sigma}_y^2\boldsymbol{I})\Big) - \tfrac{1}{2\boldsymbol{\sigma}_x^2}\|\mathrm{decoder}(\boldsymbol{z})-\boldsymbol{x}-\boldsymbol{\delta}\|_2^2 + C\Big]\\ &\leqslant -\mathrm{KL}\Big(\mathcal{N}(\boldsymbol{\mu}_y,\tilde{\boldsymbol{\sigma}}^2\boldsymbol{I}).||\mathcal{N}(\boldsymbol{\mu}_y,\boldsymbol{\sigma}_y^2\boldsymbol{I})\Big) + C - \tfrac{1}{2\boldsymbol{\sigma}_x^2}\Big(d_y^2+\epsilon^2 - 2\boldsymbol{\delta}^T(\mathrm{decoder}(\boldsymbol{z})-\boldsymbol{x})\Big)\\ &\leqslant -\mathrm{KL}\Big(\mathcal{N}(\boldsymbol{\mu}_y,\tilde{\boldsymbol{\sigma}}^2\boldsymbol{I}).||\mathcal{N}(\boldsymbol{\mu}_y,\boldsymbol{\sigma}_y^2\boldsymbol{I})\Big) + C - \tfrac{1}{2\boldsymbol{\sigma}_x^2}(d_y^2+\epsilon^2-2\epsilon d_y).\end{aligned} \tag{39}$$

where $d_y = \|\mathrm{decoder}(\boldsymbol{z})-\boldsymbol{x}\|_2, \boldsymbol{z}\sim\mathcal{N}(\boldsymbol{\mu}_y,\boldsymbol{\sigma}_y^2\boldsymbol{I})$. Now we can find ϵ for a given image $\boldsymbol{x}$ by equating Eqs. (38) and (39).

$$\epsilon \geqslant \frac{d_y^2 + 2\boldsymbol{\sigma}_x^2\Big(\mathcal{L}_{y_T}(\boldsymbol{x}) + \mathrm{KL}\Big(\mathcal{N}(\boldsymbol{\mu}_y,\tilde{\boldsymbol{\sigma}}^2\boldsymbol{I}).||\mathcal{N}(\boldsymbol{\mu}_y,\boldsymbol{\sigma}_y^2\boldsymbol{I})\Big)\Big)}{2(d_y+d_{y_T})}, \tag{40}$$

where $d_{y_T} = \|\mathrm{decoder}(\boldsymbol{z})-\boldsymbol{x}\|_2, \boldsymbol{z}\sim\mathcal{N}(\boldsymbol{\mu}_{y_T},\boldsymbol{\sigma}_{y_T}^2\boldsymbol{I})$. And we want to find:

$$\epsilon_{\boldsymbol{x}} = \min_{y\neq y_T}\epsilon. \tag{41}$$

Note that one assumption we make is that we have found the global minimum of d_{y_T} and in practice a tight estimate of the global minimum can be found.

4. Experiments

In this section, we test our model and deep convolutional neural network architectures applied to MNIST, Fashion-MNIST, and CIFAR-10 image classification datasets. We consider the following DNN architectures with corresponding dataset:

- *MNIST (Fashion-MNIST):* We trained three models, a surrogate model and two target models. The surrogate model is LeNet convolutional neural network architecture with a two-layer fully connected network and two convolutional layers. One of the target models is also LeNet architecture. For our model, we use a LeNet architecture as an encoder and a reverse LeNet architecture as a decoder, and we use $\dim(\boldsymbol{z}) = 20$for the classifier.
- *CIFAR-10:* Since the robustness properties of MNIST models may not reach to natural images, we further consider performing the same evaluation on CIFAR-10. Because fully generative classifiers are less satisfactory for classifying clean CIFAR-10 images, the clean test set accuracy for our model on CIFAR-10 is 81.51%, better than conditional PixelCNN++ [26] which achieves 72.4% clean test accuracy, and a little less than flow-based generative classifier which achieves 83.2% clean test accuracy [8]. Then we choose to work with CIFAR-2, a binary classification dataset containing "airplane" and "car" images from CIFAR-10. We also trained three models on this dataset. Two of them (including the surrogate model) are VGG-16 convolutional neural network for image classification. For our model, we use a VGG-16 architecture as an encoder and a reverse VGG-16 architecture as a decoder, and we use $\dim(\boldsymbol{z}) = 10$ for the classifier.

4.1. Classification accuracy

We use ReLU non-linearity in our network and Adam optimizer for training. The main accuracy results are shown in Table 1. Our model gets state-of-the-art on MNIST, Fashion-MNIST, and CIFAR-2 datasets. However, for high-dimension dataset, how obtaining strong classification accuracy without harming likelihood estimation for generative models is still a challenging problem [8]. For CIFAR-10 dataset, although the accuracy of our model is 10% less than CNNs', it is better than conditional PixelCNN++, and a little less than flow-based generative classifier which achieves state-of-the-art.

4.2. Generative evaluation

Since our algorithm depends upon the reconstruction error between the generated and the original images, we now show a few randomly chosen images generated by the network corresponding to test samples of different classes from three datasets in Fig. 2.

4.3. Robustness analysis

In this part, we utilize three types of adversarial examples, ℓ_2, ℓ_∞, and ℓ_0 transfer-based attacks, to test the robustness of our model. In [21,22], the authors leveraged a dataset augmentation technique to train the substitute model, which kept the same distribution of surrogate and target models. In this paper, we randomly divide the training data into two parts equally. One part is used to train the surrogate model, and the other part is used to train the target models, which guarantees that surrogate and target models are trained with data of the same distribution. Some adversarial examples of all attack methods on three datasets are shown in Fig. 3. All experimental results are shown in Tables 2 (MNIST), 3 (Fashion-MNIST), and 4 (Cifar-2). The following takes MNIST as an example to analyze the robustness of our model.

4.3.1. ℓ_∞ attacks

For ℓ_∞ transfer attacks, we craft adversarial examples by FGSM and C&W ℓ_∞ methods.

(a). FGSM. In order to improve efficiency, we use iterative FSGM to craft adversarial examples and we pick the number of iterations to be 12. As seen in Table 2, the accuracy of surrogate model reaches to 5.44% when $\epsilon$is approximately equal to 0.015. And the accuracy reduces to 13.48% of CNN-2. However, the accuracy is still over 72%for our model.

(b). C&W ℓ_∞. We report in Table 2 that the accuracy of surrogate model and target models by C&W ℓ_∞ adversarial attacks. It shows that C&W ℓ_∞ is a strong attack. The accuracy of surrogate model is 0.0% by C&W ℓ_∞ attack. However, for target models, our model has better robustness than CNN-2 while the accuracy of our model is 72.15% and the CNN-2 is 58.76%.

In conclusion, our model has got better robustness on ℓ_∞ transfer attacks.

Table 1
The main accuracy results of all models.

	Model		
Dataset	CNN-1	CNN-2	Our Model
MNIST	99.2%	99.2%	99.1%
Fashion-MNIST	91.29%	91.26%	91.47%
CIRFAR-10	92.17%	91.42%	81.51%
CIRFAR-2	90.1%	91.1%	90.25%

(a) Generated

(b) original

Fig. 2. Generated and original images from MNIST, Fashion-MNIST, and CIFAR-10 datasets.

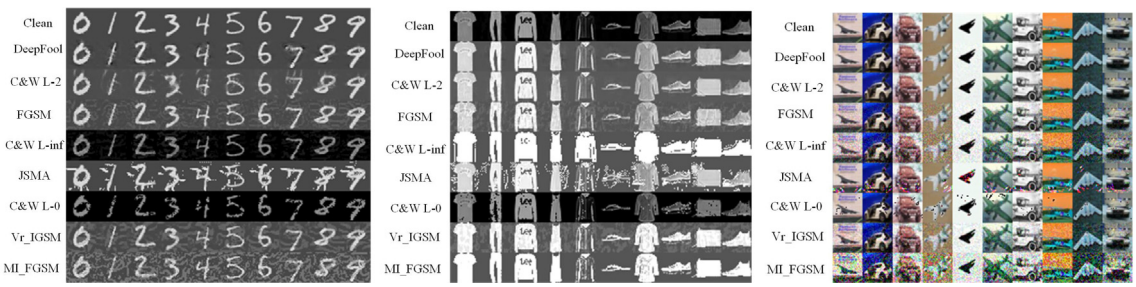

Fig. 3. Clean data and adversarial examples of all attack methods on MNIST, Fashion-MNIST, and CIFAR-2 datasets.

Table 2
Performance comparisons of transfer attacks on MNIST.

		Surrogate model (CNN1)	Target model (CNN2)	Target model (Our model)
	Clean	98.59%	98.51%	98.27%
L_∞	FGSM	1.12%	13.48%	**72.23%**
	CW_L_∞	0.00%	58.60%	**75.40%**
L_2	Deepfool	0.00%	83.85%	**95.71%**
	CW_L_2	0.46%	75.95%	**94.34%**
L_0	JSMA	12.08%	27.86%	**48.87%**
	CW_L_0	0.00%	64.04%	**76.82%**
Strong	Vr-IGSM	2.72%	65.42%	**85.13%**
	MI-FGSM	2.00%	29.29%	**79.38%**

Table 3
Performance comparisons of transfer attacks on Fashion-MNIST.

		Surrogate model (CNN1)	Target model (CNN2)	Target model (Our model)
	Clean	89.44%	89.59%	88.25%
L_∞	FGSM	5.44%	12.50%	**60.42%**
	CW_L_∞	0.00%	58.76%	**72.15%**
L_2	Deepfool	0.00%	77.45%	**85.08%**
	CW_L_2	1.48%	62.66%	**82.14%**
L_0	JSMA	10.56%	30.50%	**44.26%**
	CW_L_0	0.00%	64.44%	**71.81%**
Strong	Vr-IGSM	1.82%	56.49%	**77.45%**
	MI-FGSM	1.60%	26.55%	**72.28%**

Table 4
Performance comparisons of transfer attacks on Cifar-2.

		Surrogate model (CNN1)	Target model (CNN2)	Target model (Our model)
	Clean	86.68%	86.42%	85.88%
L_∞	FGSM	1.48%	62.50%	**80.47%**
	CW_L_∞	7.00%	48.02%	**62.18%**
L_2	Deepfool	0.00%	80.66%	**83.92%**
	CW_L_2	0.00%	81.64%	**82.03%**
L_0	JSMA	5.20%	76.81%	**83.74%**
	CW_L_0	0.00%	77.44%	**81.31%**
Strong	Vr-IGSM	1.76%	60.39%	**73.40%**
	MI-FGSM	1.60	55.68%	**68.30%**

4.3.2. ℓ_2 attacks

For ℓ_2 transfer attacks, we craft adversarial examples by DeepFool and C&W ℓ_2 methods.

(a). DeepFool. We report in Table 2 that the accuracy of surrogate model and target models by DeepFool adversarial attacks. It shows that DeepFool is an effective attack that the perturbations are too small to perceptible. The accuracy of surrogate model is 0.0% by DeepFool attack. The transfer attack on CNN-2 and our model has no significant effect due to the small disturbance of adversarial examples. However, for target models, our model has better robustness than CNN-2 while the accuracy of our model is 95.71% and the CNN-2 is 83.85%.

(b). C&W ℓ_2. We report in Table 2 that the accuracy of surrogate model and target models by C&W ℓ_2 adversarial attacks. The accuracy of surrogate model is 0.0% by C&W ℓ_2 attack. However, for target models, our model has better robustness than CNN-2 while the accuracy of our model is 94.34% and the CNN-2 is 75.95%.

In conclusion, our model has got better robustness on ℓ_2 transfer attacks.

4.3.3. ℓ_0 attacks

For ℓ_0 transfer attacks, we generate adversarial examples by JSMA and C&W ℓ_0 methods.

(a). JSMA. We report in Table 2 that the accuracy of surrogate model and target models by JSMA adversarial attacks. It shows that JSMA is an effective attack. The CNN-1 only gets the accuracy of 12.08% on adversarial examples generating by JSMA attack. The transfer attack on CNN-2 and our model is significant successful because the decreasing of accuracy results due to the large disturbance. However, our model also has better robutness than CNN-2. The accuracy of our model is 48.87% while the CNN-2 is 27.86%.

(b). C&W ℓ_0. We report in Table 2 that the accuracy of surrogate model and target models by C&W ℓ_0 adversarial attacks. It shows that C&W ℓ_0 is an effective attack. The surrogate model only gets the accuracy of 0.0% on adversarial examples generating by C&W ℓ_0 attack. The transfer attack on CNN-2 and our model is significant work because the decreasing of accuracy results. However, for target models, our model has better robustness than CNN-2 while the accuracy of our model is 76.82% and the CNN-2 is 64.04%.

In conclusion, our model has got better robustness on ℓ_0 attacks.

4.3.4. Strong transfer-based attacks

To further test the robustness of our model, we take two strong transfer attacks, Vr-IGSM and MI-FGSM, to attack the target models. As shown in Table 2, the accuracy of surrogate model on these two attacks is 1.82%and 1.60%, which indicates

Table 6
Performance comparisons with other defense methods against transfer attacks on Fashion-MNIST

	No defense	Adv Training	ComDefend	Our method	Our method + Adv Training	Our method + ComDefend
Clean	91.3%	90.4%	91.3%	91.5%	90.1%	91.5%
FGSM	45.0%	**83.5%**	66.4%	64.3%	81.1%	83.3%
CW_L_∞	60.9%	61.3%	75.6%	74.9%	77.6%	**80.1%**
Deepfool	77.5%	88.0%	83.7%	90.8%	89.6%	**91.2%**
CW_L_2	64.6%	85.2%	79.1%	85.0%	87.4%	**87.7%**
JSMA	31.8%	32.9%	40.6%	44.9%	43.8%	**64.7%**
CW_L_0	66.6%	79.7%	80.3%	73.1%	78.7%	**82.2%**
Vr-IGSM	57.5%	**88.0%**	71.1%	76.4%	80.2%	78.8%
MI-FGSM	28.6%	82.6%	68.8%	75.2%	**84.3%**	75.5%

Table 7
Performance comparisons with other defense methods against transfer attacks on Cifar-2

	No defense	Adv Training	ComDefend	Our method	Our method + Adv Training	Our method + ComDefend
Clean	91.1%	89.7%	91.1%	90.3%	89.2%	90.3%
FGSM	69.6%	**82.5%**	75.8%	80.1%	81.1%	82.0%
CW_L_∞	50.0%	60.1%	70.3%	62.4%	72.2%	**73.3%**
Deepfool	84.7%	85.0%	88.7%	**89.6%**	88.3%	89.6%
CW_L_2	85.6%	85.8%	87.8%	**89.6%**	87.3%	89.6%
JSMA	80.2%	80.6%	85.4%	86.5%	80.1%	**86.8%**
CW_L_0	79.3%	80.4%	84.1%	83.7%	84.4%	**85.5%**
Vr-IGSM	62.4%	82.1%	76.8%	75.6%	**82.2%**	79.7%
MI-FGSM	57.7%	**79.2%**	74.9%	71.4%	79.0%	76.5%

Table 5
Performance comparisons with other defense methods against transfer attacks on MNIST

	No defense	Adv Training	ComDefend	Our method	Our method + Adv Training	Our method + ComDefend
Clean	99.2%	98.2%	99.2%	99.1%	99.0%	99.2%
FGSM	49.6%	**92.6%**	78.5%	81.6%	90.7%	90.3%
CW_L_∞	60.5%	60.4%	86.6%	82.4%	86.9%	**88.7%**
Deepfool	88.2%	97.8%	93.2%	98.0%	96.2%	**98.3%**
CW_L_2	71.4%	95.4%	85.8%	94.9%	**96.5%**	96.4%
JSMA	37.0%	28.3%	44.7%	57.8%	52.5%	**69.8%**
CW_L_0	58.7%	81.7%	84.2%	73.8%	82.4%	**86.1%**
Vr-IGSM	65.4%	**96.2%**	79.6%	85.1%	89.0%	92.4%
MI-FGSM	29.3%	91.4%	74.0%	79.4%	**91.7%**	81.9%

the effectiveness of the two attacks. However, for target models, our model has better robustness than CNN-2 while the accuracy of our model is $85.13\%, 79.38\%$ and the CNN-2 is $65.42\%, 29.29\%$.

4.3.5. Compare with other defense methods

As shown in Table 5, we compare our approach with other effective defense methods on MNIST dataset. All models are trained by the whole training dataset. Although adversarial training (based on FGSM) and ComDefend have achieved better results in defending against some transfer-based attacks, our method is better at defending against DeepFool and JSMA transfer attacks. In addition, our method, based on a generative model, can be combined with adversarial training or Comdefend, which can greatly improve the performance of defense. This is a big advantage for our method. Table 6-7.

4.4. Summary

In summary, our model reaches state-of-the-art on MNIST and Fashion-MNIST datasets. For CIFAR-10 dataset, although the accuracy of our model is below the deep discriminative models, it is above the generative classification models (such as PixelCNN++). In addition, our model is more robust to ℓ_2, ℓ_0, and ℓ_∞ transfer-based attacks than the discriminative models. Compared with other high performance defense methods, our method is competitive. Besides, our model can be combined with adversarial training or Comdefend to achieve better defense against transfer-based attacks.

5. Conclusion

In this work, a novel generative classifier with Gaussian mixture prior on latent embedding based on VAEs has been proposed. Our model simulates the generating process of data in a natural way. We have given proof that our generative clas-

sifier is more robust to many recent transfer-based attacks while our model can get state-of-the-art and generate high quality images.

For large datasets, although the generative models can effectively learn from the data, these models also tend to make stronger assumptions about the data than discriminative models that are treated purely differently. When the models are wrong, they usually lead to higher asymptotic bias. Therefore, finding a good generative model is a meaningful exploration. And using generation classifiers provides an interesting way to evaluate the ability to build models and improve their capacity to process high-dimensional data sets. In general, we believe that the advances in generating classifiers can better motivate the design of attack, defense, and detection technologies.

CRediT authorship contribution statement

Chen Zhang: Conceptualization, Methodology, Software, Writing - original draft, Formal analysis. **Zhuo Tang:** Writing - review & editing, Resources, Project administration. **Youfei Zuo:** Visualization. **Kenli Li:** Supervision, Resources. **Keqin Li:** Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] A. Athalye, N. Carlini, D. Wagner, Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. arXiv preprint arXiv:1802.00420, 2018..
[2] N. Carlini, D. Wagner, Towards evaluating the robustness of neural networks, in: 2017 IEEE Symposium on Security and Privacy (SP), IEEE, 2017, pp. 39–57.
[3] S. Cheng, Y. Dong, T. Pang, H. Su, J. Zhu, Improving black-box adversarial attacks with a transfer-based prior, Adv. Neural Inform. Process. Syst. (2019) 10932–10942.
[4] M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, N. Usunier, Parseval networks: Improving robustness to adversarial examples, in: Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org, 2017. pp. 854–863..
[5] G.S. Dhillon, K. Azizzadenesheli, Z.C. Lipton, J. Bernstein, J. Kossaifi, A. Khanna, A. Anandkumar, Stochastic activation pruning for robust adversarial defense. arXiv preprint arXiv:1803.01442, 2018..
[6] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, J. Li, Boosting adversarial attacks with momentum, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 9185–9193.
[7] R. Feinman, R.R. Curtin, S. Shintre, A.B. Gardner, Detecting adversarial samples from artifacts. arXiv preprint arXiv:1703.00410, 2017..
[8] E. Fetaya, J. Jacobsen, W. Grathwohl, R.S., Zemel, Understanding the limitations of conditional generative models, in: International Conference on Learning Representations (ICLR 2020), 2020..
[9] I.J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572, 2014..
[10] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE international conference on computer vision, 2015, pp. 1026–1034.
[11] X. Jia, X. Wei, X. Cao, H. Foroosh, Comdefend: An efficient image compression model to defend adversarial examples, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 6084–6092.
[12] D.P. Kingma, M. Welling, Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013..
[13] A. Kurakin, I. Goodfellow, S. Bengio, Adversarial machine learning at scale. arXiv preprint arXiv:1611.01236, 2016..
[14] A. Kurakin, I. Goodfellow, S. Bengio, Y. Dong, F. Liao, M. Liang, T. Pang, J. Zhu, X. Hu, C. Xie, C., et al., Adversarial attacks and defences competition, in: The NIPS'17 Competition: Building Intelligent Systems. Springer, 2018. pp. 195–231..
[15] X. Li, J. She, Collaborative variational autoencoder for recommender systems, in: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, ACM, 2017, pp. 305–314.
[16] Y. Li, J. Bradshaw, Y. Sharma, Are generative classifiers more robust to adversarial attacks?, International Conference on Machine Learning (ICML), 2019, pp 3804–3814.
[17] Y. Li, L. Li, L. Wang, T. Zhang, B. Gong, Nattack: Learning the distributions of adversarial examples for an improved black-box attack on deep neural networks. arXiv preprint arXiv:1905.00441, 2019b..
[18] Y. Liu, X. Chen, C. Liu, D. Song, Delving into transferable adversarial examples and black-box attacks. Fifth International Conference on Learning Representations (ICLR 2017), 2017..
[19] J.H. Metzen, T. Genewein, V. Fischer, B. Bischoff, On detecting adversarial perturbations. arXiv preprint arXiv:1702.04267, 2017..
[20] S.M. Moosavi-Dezfooli, A. Fawzi, P. Frossard, Deepfool: a simple and accurate method to fool deep neural networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2574–2582.
[21] N. Papernot, P. Mcdaniel, I. Goodfellow, Transferability in machine learning: from phenomena to black-box attacks using adversarial samples, Cryptography and Security, 2016.
[22] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z.B. Celik, A. Swami, Practical black-box attacks against machine learning, in: Proceedings of the 2017 ACM on Asia conference on computer and communications security, 2017, pp. 506–519.
[23] 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 (EuroS&P), IEEE, 2016, pp. 372–387.
[24] N. Papernot, P. McDaniel, X. Wu, S. Jha, A. Swami, Distillation as a defense to adversarial perturbations against deep neural networks, in: in: 2016 IEEE Symposium on Security and Privacy (SP), IEEE, 2016, pp. 582–597.
[25] A. Razavi, A. van den Oord, O. Vinyals, Generating diverse high-fidelity images with vq-vae-2, Adv. Neural Inform. Process. Syst. (2019) 14837–14847.
[26] T. Salimans, A. Karpathy, X. Chen, D.P. Kingma, Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. arXiv preprint arXiv:1701.05517, 2017..
[27] L. Schott, J. Rauber, M. Bethge, W. Brendel, Towards the first adversarially robust neural network model on mnist, in: Seventh International Conference on Learning Representations (ICLR 2019), 2019. pp. 1–16..
[28] C. Sitawarin, D. Wagner, Defending against adversarial examples with k-nearest neighbor. arXiv preprint arXiv:1906.09525, 2019..
[29] L. Smith, Y. Gal, Understanding measures of uncertainty for adversarial example detection. arXiv preprint arXiv:1803.08533, 2018..

[30] Y. Song, T. Kim, S. Nowozin, S. Ermon, N. Kushman, Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. arXiv preprint arXiv:1710.10766, 2017..
[31] D. Su, H. Zhang, H. Chen, J. Yi, P.Y. Chen, Y. Gao, Is robustness the cost of accuracy?–a comprehensive study on the robustness of 18 deep image classification models, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 631–648.
[32] P. Sun, D.A. Moses, E.F. Chang, Modeling neural dynamics during speech production using a state space variational autoencoder, in: 2019 9th International IEEE/EMBS Conference on Neural Engineering (NER), IEEE, 2019, pp. 428–432.
[33] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199, 2013..
[34] M. Tan, Q.V. Le, Efficientnet: Rethinking model scaling for convolutional neural networks. arXiv preprint arXiv:1905.11946, 2019..
[35] H. Touvron, A. Vedaldi, M. Douze, H. Jégou, Fixing the train-test resolution discrepancy, Adv. Neural Inform. Process. Syst. (2019) 8250–8260.
[36] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, P. McDaniel, Ensemble adversarial training: Attacks and defenses. arXiv preprint arXiv:1705.07204, 2017..
[37] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, A. Madry, Robustness may be at odds with accuracy. arXiv preprint arXiv:1805.12152, 2018..
[38] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, X. Tang, Residual attention network for image classification, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 3156–3164.
[39] W. Wang, Z. Gan, H. Xu, R. Zhang, G. Wang, D. Shen, C. Chen, L. Carin, Topic-guided variational autoencoders for text generation. arXiv preprint arXiv:1903.07137, 2019..
[40] E. Wong, L. Rice, J.Z. Kolter, Fast is better than free: Revisiting adversarial training, in: International Conference on Learning Representations (ICLR 2020), 2020..
[41] L. Wu, Z. Zhu, C. Tai, et al., Understanding and enhancing the transferability of adversarial examples. arXiv preprint arXiv:1802.09707, 2018..
[42] W. Xu, H. Sun, C. Deng, Y. Tan, Variational autoencoder for semi-supervised text classification, Thirty-First AAAI Conference on Artificial Intelligence, 2017, pp. 3358–3364.