# Regularization with Latent Space Virtual Adversarial Training

Genki Osada[1,2,5], Budrul Ahsan[1,3], Revoti Prasad Bora[4] *, and Takashi Nishide[2]

[1] Philips Co-Creation Center, Japan
[2] University of Tsukuba, Japan
[3] The Tokyo Foundation for Policy Research, Japan
[4] Philips India Limited
[5] I Dragon Corporation, Japan
homerunrun@hotmail.com

**Abstract.** Virtual Adversarial Training (VAT) has shown impressive results among recently developed regularization methods called consistency regularization. VAT utilizes adversarial samples, generated by injecting perturbation in the input space, for training and thereby enhances the generalization ability of a classifier. However, such adversarial samples can be generated only within a very small area around the input data point, which limits the adversarial effectiveness of such samples. To address this problem we propose LVAT (Latent space VAT), which injects perturbation in the latent space instead of the input space. LVAT can generate adversarial samples flexibly, resulting in more adverse effect and thus more effective regularization. The latent space is built by a generative model, and in this paper we examine two different type of models: variational auto-encoder and normalizing flow, specifically Glow.
We evaluated the performance of our method in both supervised and semi-supervised learning scenarios for an image classification task using SVHN and CIFAR-10 datasets. In our evaluation, we found that our method outperforms VAT and other state-of-the-art methods.

**Keywords:** consistency regularization, adversarial training, image classification, semi-supervised learning, and unsupervised learning.

## 1 Introduction

One of the goals of training machine learning models is to avoid overfitting. To address overfitting, we use various regularization techniques. Recently *consistency regularization* has shown remarkable results to overcome overfitting problems in deep neural networks. Consistency regularization is also known as perturbation-based methods, and its basic strategy is to perturb inputs during the learning process and force the model to be robust against them. Perturbation is defined as randomizing operations such as dropout, introducing Gaussian noise, and data augmentation. Consistency regularization is achieved by

*Currently working at Lowe's Services India Pvt. Ltd.

introducing a regularization term, called consistency cost. Consistency cost is designed to penalize the discrepancy between the model outputs, with and without perturbations. Thus consistency regularization can work without class labels, which leads to two attractive features: 1) it can extend most supervised learning methods to semi-supervised learning methods, which enables us to use unlabeled data for the model training, and 2) like the techniques such as dropout and data augmentation, it can be applied to most existing models without modifying the model architecture.

Among the consistency regularization methods, Virtual Adversarial Training (VAT) [25, 26] has shown promising results. The way of generating perturbation in VAT is different from other consistency regularization methods. Perturbation in VAT is not random, instead deliberately generated towards a direction that causes the most adverse effects on the model output. However, perturbations in VAT can work only a very small area around the input data points. It is because those perturbations are generated based on the local sensitivity, i.e., the gradients of the model outputs w.r.t. the tiny shift in the input space (Section 3.1). We focus on the fact that such *local constraint* limits the offensive power of adversarial perturbations, and thus hinders the effectiveness of VAT as a consistency regularization. Based on this, we aim to overcome this limitation and develop a method to generate perturbation more flexibly, which would lead to better generalization.

In this paper, we propose VAT based consistency regularization that utilizes the latent space. The key idea underlying our method is transforming the space in which the computation of perturbation is performed. In our method, the input data is mapped to the latent space, and we compute perturbation and inject it into the point in the latent space. Then, adversarial examples are generated by mapping such perturbed latent point back to the point in the input space, and therefore, there is no local constraint which hinders VAT effectiveness. To map to and from the latent space, we use a generative model, which we call *transformer*, and in this paper we examine two models: variational auto-encoder (VAE) [16] and normalizing flow, specifically, Glow [17]. We used SVHN and CIFAR-10 datasets, which are common benchmarking datasets, to demonstrate that our method improves VAT and outperforms other state-of-the-art methods. To the best of our knowledge, this work is the first to introduce the latent space in the context of consistency regularization.

*Notation.* We consider a classification task. Suppose we have $K$ classes, taking an input $\mathbf{x}_i \in \mathcal{X}$, we want to predict the class label $y_i \in \{1, 2, \ldots, K\}$[6], where $\mathcal{X}$ is a sample space with data points. Our purpose is to learn $K$-class classifier model $f : \mathcal{X} \mapsto \mathbb{R}^K$ parameterized by $\theta$, where $\mathbb{R}$ denotes real numbers, the $k$-th element $f(\mathbf{x})_k$ is called logit for class $k$, and $y_{\text{pred}} = \arg\max_k f(\mathbf{x})_k \in \{1, 2, \ldots, K\}$ corresponds to the model prediction of the class. Also, labeled dataset is denoted by $D_{\text{l}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_{\text{l}}}$ with $N_{\text{l}}$ samples and unlabeled dataset is denoted by $D_{\text{u}} = \{\mathbf{x}_i\}_{i=1}^{N_{\text{u}}}$ with $N_{\text{u}}$ samples.

[6] We write scalars and vectors by non-bold and bold letters, respectively.

## 2 Related Work

We focus mainly on state-of-the-art methods that are related to our approach. These can be broadly categorized as follows: consistency regularization, graph-based methods, and GAN-based methods.

**Consistency Regularization.** The VAT and our proposed method LVAT belong to this category. The assumption underlying these methods is *local consistency* [37]: nearby points in the input space are likely to have the same output. In general, the model predictions for the points near the decision boundaries are sensitive to perturbations and prone to be misclassified by being perturbed. To mitigate this sensitivity, this type of method employs the regularizing loss function, called consistency cost, which aims to train the model so that its outputs would be consistent for the inputs both with and without perturbation. Because the consistency cost becomes large at the points near the decision boundaries, the regularizing effect works so that the decision boundary would be kept far away from such points, which leads to better generalization in testing time. As the simplest case, the Π-Model [20] employs the following consistency cost $R(\mathbf{x})$:

$$R(\mathbf{x}) := \|f(\tilde{\mathbf{x}}_1, \theta) - f(\tilde{\mathbf{x}}_2, \theta)\|_2^2 \tag{1}$$

$$\tilde{\mathbf{x}}_1 \sim \mathrm{Perturb}(\mathbf{x}),\ \tilde{\mathbf{x}}_2 \sim \mathrm{Perturb}(\mathbf{x}) \tag{2}$$

where $\|\cdot\|_2$ denotes $L_2$ norm, and $\mathrm{Perturb}(\cdot)$ is a function that applies a stochastic deformation (i.e., data augmentation), random noise addition, and dropout, thus outputting different $\tilde{\mathbf{x}}_i$ each time.

Dropout, Gaussian noise, and randomized data augmentation have been chosen as perturbations in [1, 2, 20, 28, 31, 34]. [36] has shown that the latest data augmentation techniques, AutoAugment and Cutout, were quite effective to use as perturbations. Although in the Π-Model, $\mathrm{Perturb}(\cdot)$ is applied to both of $\mathbf{x}$ in $R(\mathbf{x})$, $\mathrm{Perturb}(\cdot)$ is applied only to one $\mathbf{x}$ in $R(\mathbf{x})$ in VAT and LVAT. Furthermore, perturbations used in VAT and LVAT are not random but are carefully computed, as we describe it in the next section.

**Graph-based Methods.** In contrast to the above consistency regularization methods, graph-based methods assume *global consistency* [37]: all samples that map to the same class label should belong to a single cluster. [14] proposed a method that captures a structure of samples within a mini-batch by means of label propagation, and then forces samples belonging to the same class to form compact clusters in the feature space. Smooth Neighbors on Teacher Graphs (SNTG) [23] computes unsupervised loss function for each mini-batch, in which attraction force between samples belonging to the same class and repulsion force between samples belonging to the different classes are realized. Consistency regularizations focus on the sensitivity of each data point to perturbations, whereas graph-based methods regularize a whole structure of data points within a mini-batch. Graph-based methods and consistency regularizations are not mutually

exclusive, but rather complementary. These two could be implemented at the same time, and in fact [23] reported that combining with SNTG steadily improved the performance of all consistency regularization methods in their experiments.

**GAN-based Methods.** Several works have utilized the samples generated in GAN framework as another kind of perturbation injection [8, 21, 32, 33]. Among them, BadGAN [5] presented impressive performance. As opposed to the usual GAN framework, the generator in BadGAN generates unrealistic samples, which can be viewed as a data augmentation targeting the lower density regions in the given data distribution. With such data augmentation, BadGAN aims to draw better decision boundaries, and thus their objective is similar to VAT and our proposed method. We compare our method with these methods in result section.

**Adversarial Examples for Generative Models.** We finally note that, in the filed of adversarial machine learning, there are studies in regard to the latent space of generative models. However, their objectives are not regularization like ours. [3] studied attacking the generative models, and [9] derived a fundamental upper bound on robustness against adversarial perturbations. Our goal is to achieve better consistency regularization, and to this end, we aim to generate more effective adversarial examples by utilizing the latent space.

## 3 Background

### 3.1 Virtual Adversarial Training and Local Constraint

Perturbations in VAT are deliberately generated so that its direction could cause the most adverse effects on the model outputs, i.e., classification predictions. Formally, letting

$$R(\mathbf{x}, \mathbf{r}) := \mathrm{KL}(f(\mathbf{x}, \theta) \parallel f(\mathbf{x} + \mathbf{r}, \theta)), \tag{3}$$

adversarial perturbation $\mathbf{r}_{\mathrm{vat}}$ is defined as

$$\mathbf{r}_{\mathrm{vat}} := \arg\max_{\mathbf{r}} \{R(\mathbf{x}, \mathbf{r}); \|\mathbf{r}\|_2 \leq \epsilon_{\mathrm{vat}}\} \tag{4}$$

where $\mathrm{KL}(p \parallel q)$ denotes KullbackLeibler (KL) divergence between distributions $p$ and $q$, and $\epsilon_{\mathrm{vat}}$ is a hyper-parameter to decide the magnitude of $\mathbf{r}_{\mathrm{vat}}$[7]. Once $\mathbf{r}_{\mathrm{vat}}$ is computed, the consistency cost is given as:

$$\begin{aligned} L_{\mathrm{vat}} &:= R(\mathbf{x}, \mathbf{r}_{\mathrm{vat}}) \\ &= \mathrm{KL}(f(\mathbf{x}, \theta) \parallel f(\mathbf{x} + \mathbf{r}_{\mathrm{vat}}, \theta)). \end{aligned} \tag{5}$$

[7]We use the suffix of 'vat' to distinguish from the symbols that will be used later in the description of our proposed method.

This regularizing cost encourages the classifier to be trained so that it outputs consistent predictions for the clean input $\mathbf{x}$ and the adversarially perturbed input $\mathbf{x} + \mathbf{r}_{\text{vat}}$.

Eq. (4) can be rewritten as $\mathbf{r}_{\text{vat}} = \epsilon_{\text{vat}}\mathbf{u}$, where $\mathbf{u}$ is a unit vector in the same direction as $\mathbf{r}_{\text{vat}}$ and the maximum magnitude of $\mathbf{r}_{\text{vat}}$ is given by $\epsilon_{\text{vat}}$. To calculate $\mathbf{u}$, [26] has presented following fast approximation method. Under the assumption that $f(\mathbf{x}, \theta)$ is twice differentiable with respect to $\theta$, the second-order Taylor expansion around the point of $\mathbf{r} = 0$ yields

$$R(\mathbf{x}, \mathbf{r}) \approx R(\mathbf{x}, 0) + R'(\mathbf{x}, 0)\mathbf{r} + \frac{1}{2}\mathbf{r}^T H\mathbf{r} \tag{6}$$

$$= \frac{1}{2}\mathbf{r}^T H\mathbf{r} \tag{7}$$

where $H$ is the Hessian matrix given by $H := R''(\mathbf{x}, 0)$, and $R(\mathbf{x}, 0)$ and $R'(\mathbf{x}, 0)$ in the first line are zeros since $\mathrm{KL}(p \parallel q)$ takes the minimal value zero when $p = q$, i.e., $\mathbf{r} = 0$. Thus, taking the eigenvector of $H$ which has the largest eigenvalue is required to solve Eq. (4). To reduce the computational cost, a finite difference power method is introduced. Given a random unit vector $\mathbf{d}$, the iterative calculation of $\mathbf{d} \leftarrow \overline{H\mathbf{d}}$ where $\overline{H\mathbf{d}} := H\mathbf{d}/\|H\mathbf{d}\|_2$, makes the $\mathbf{d}$ converge to $\mathbf{u}$. With a small constant $\xi$, finite difference approximation follows as

$$H \approx \left(R'(\mathbf{x}, 0 + \xi\mathbf{d}) - R'(\mathbf{x}, 0)\right)/\xi\mathbf{d} \tag{8}$$

$$Hd = R'(\mathbf{x}, \xi\mathbf{d})/\xi \tag{9}$$

where we use the fact that $R'(\mathbf{x}, 0) = 0$. Then the repeated application of $\mathbf{d} \leftarrow \overline{R'(\mathbf{x}, \xi\mathbf{d})}$ yields $\mathbf{u}$. [26] reported that sufficient result was reached by only one iteration. As a result, with a given $\epsilon_{\text{vat}}$, Eq. (4) can be computed as:

$$\mathbf{r}_{\text{vat}} = \epsilon_{\text{vat}}\overline{R'(\mathbf{x}, \xi\mathbf{d})}. \tag{10}$$

The pseudo-code describing the computation of $L_{\text{vat}}$ defined in Eq. (5) is shown in Algorithm 1, which will be helpful to clarify in which part our proposed method differs from the VAT.

*Local Constraint.* As we can see in Eq. (6), VAT algorithm works under the assumption that $\mathbf{r}$ is very small such that Taylor expansion is applicable. This means that adversarial examples $\mathbf{x} + \mathbf{r}_{\text{vat}}$ are crafted as only a very small shift from $\mathbf{x}$, which hinders the search for more adverse examples. Our purpose is to remove this constraint to generate adversarial examples flexibly, and to this end, we compute $\mathbf{r}_{\text{vat}}$ in the latent space.

### 3.2 Transformer

To map to and from the latent space, we use a generative model. In this paper, we examine two types of model, VAE and normalizing flow. The VAE is approximate inference and the dimensionality of the latent space is usually much smaller than

that of the input space. On the other hand, the normalizing flow is exact inference and the dimensionality of the latent space is kept equal to that of the input space, i.e., lossless conversion. We will use the term *transformer* as a generic name to refer to these two.

**Variational Auto-Encoder** consists of two networks: the encoder (Enc) that maps a data sample $\mathbf{x}$ to $\mathbf{z}$ in latent space, and the decoder (Dec) that maps $\mathbf{z}$ back to a point $\hat{\mathbf{x}}$ in the input space as:

$$\mathbf{z} \sim \mathrm{Enc}(\mathbf{x}) = q(\mathbf{z}|\mathbf{x}),\ \hat{\mathbf{x}} \sim \mathrm{Dec}(\mathbf{z}) = p(\mathbf{x}|\mathbf{z}). \tag{11}$$

The VAE regularizes the encoder by imposing a prior over the latent distribution $p(\mathbf{z})$. Typically $p(\mathbf{z})$ is set as a standard normal distribution $\mathcal{N}(0, \mathbf{I})$. The VAE loss is:

$$L_{\mathrm{vae}} = -\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})}\right] \tag{12}$$

and it can be written as the sum of the following two terms: the expectation of negative log likelihood, i.e., the reconstruction error, $\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{x}|\mathbf{z})\right]$, and a prior regularization term, $\mathrm{KL}(q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}))$.

**Normalizing Flow.** Suppose $g(\cdot)$ is an invertible function and let $\mathbf{h}_0$ and $\mathbf{h}_1$ be random variables of equal dimensionality. Under the change of variables rule, transformation $\mathbf{h}_1 = g(\mathbf{h}_0)$ can be written as the change in the probability density function: $p(\mathbf{h}_0) = p(\mathbf{h}_1)|\det(d\mathbf{h}_1/d\mathbf{h}_0)|$. Stacking this transformation $L$-times as $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_L$ yields:

$$p(\mathbf{h}_0) = p(\mathbf{h}_L)\prod_{i=1}^{L} |\det(d\mathbf{h}_i/d\mathbf{h}_{i-1})|, \tag{13}$$

and taking the logarithm results in:

$$\log p(\mathbf{x}) = \log p(\mathbf{z}) + \sum_{i=1}^{L} \log |\det(d\mathbf{h}_i/d\mathbf{h}_{i-1})| \tag{14}$$

where we define $\mathbf{h}_0 := \mathbf{x}$ and $\mathbf{h}_L := \mathbf{z}$. Such a series of transformations can gradually transform $p(\mathbf{x})$ into a target distribution $p(\mathbf{z})$ of any form. Setting $p(\mathbf{z}) = \mathcal{N}(0, \mathbf{I})$ is especially called a normalizing flow [30].

As Eq. (14) is the form of log-likelihood, the learning objective is to maximize $\mathbb{E}_{p(\mathbf{x})}\log p(\mathbf{x})$ by optimizing $g(\cdot)$. The function $g(\cdot)$ must be designed to have the tractability to compute its inverse and the determinant of Jacobian matrix $|\det(d\mathbf{h}_i/d\mathbf{h}_{i-1})|$ in Eq. (14), and several methods have been proposed in this regard. *Autoregressive* models [11, 18, 27] have a powerful expression but are computationally slow due to non-parallelization. Thus, we use *split coupling* models [6, 7], specifically, Glow [17]. For brevity, we refer the reader to [17].

Similarly to the case of VAE, we denote transformation $\mathbf{x} \to \mathbf{z}$ by $\mathbf{z} = \mathrm{Enc}(\mathbf{x})$ and $\mathbf{z} \to \mathbf{x}$ by $\mathbf{x} = \mathrm{Dec}(\mathbf{z})$, and we call them just Enc() and Dec() as generic notations, for convenience.

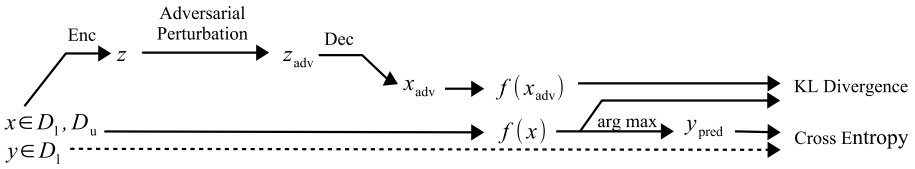



**Fig. 1.** Overview of our method. Only during training, we place trained transfomer, i.e., Enc() and Dec(), in front of classifier $f()$ being trained. While for $\mathbf{x} \in D_{\text{l}}, D_{\text{u}}$ classifier outputs $f(\mathbf{x})$, predictive class label $y_{\text{pred}}$ is produced only for $\mathbf{x} \in D_{\text{l}}$. KL divergence corresponds to the consistency const $L_{\text{lvat}}$.

## 4 Method

Our proposed method applies Eq. (4), i.e., Eq. (10), to the latent space. It means that our method generates perturbations based on the gradients of the model outputs w.r.t. the shift in the latent space, and therefore, the latent space in our method is required to be a continuous distribution. Thus, vanilla Auto-Encoder and Denoising Auto-Encoder [35], which do not construct the latent space as a continuous distribution, are out of our selection. Instead, we choose two different types of generative models, VAE and Glow, and we build those models so that the latent space $p(\mathbf{z})$ forms $\mathcal{N}(0, \mathbf{I})$. We call our proposed method *LVAT* standing for Virtual Adversarial Training in the Latent space, and we refer to LVAT using VAE and LVAT using Glow as *LVAT-VAE* and *LVAT-Glow*, respectively. In Fig. 1, we show the overview of LVAT. We deploy the transfomer in the fore stage of the classifier that we want to train. During training, by mapping the input $\mathbf{x} \in D_{\text{l}}, D_{\text{u}}$ to the latent space by Enc(), the latent representation $\mathbf{z} = \text{Enc}(\mathbf{x})$ is computed. It is followed by applying Eq. (4) to $\mathbf{z}$ and computing the adversarial perturbation in the latent space, $\mathbf{r}_{\text{lvat}}$, and the adversarial latent representation $\mathbf{z}_{\text{adv}} = \mathbf{z} + \mathbf{r}_{\text{lvat}}$ is computed. Then, by putting $\mathbf{z}_{\text{adv}}$ through Dec(), we obtain adversarial samples $\mathbf{x}_{\text{adv}} = \text{Dec}(\mathbf{z}_{\text{adv}})$. Here, we define

$$R_{\text{lvat}}(\mathbf{x}, \mathbf{r}) := \text{KL}(f(\mathbf{x}, \theta) \parallel f(\mathbf{x}', \theta)) \tag{15}$$

$$\mathbf{x}' = \text{Dec}(\text{Enc}(\mathbf{x}) + \mathbf{r}) \tag{16}$$

and the adversarial perturbation $\mathbf{r}_{\text{lvat}}$ and the consistency cost $L_{\text{lvat}}$ are defined as:

$$\mathbf{r}_{\text{lvat}} := \arg\max_{\mathbf{r}} \{R_{\text{lvat}}(\mathbf{x}, \mathbf{r}); \|\mathbf{r}\|_2 \leq \epsilon_{\text{lvat}}\} \tag{17}$$

$$\mathbf{x}_{\text{adv}} := \text{Dec}(\text{Enc}(\mathbf{x}) + \mathbf{r}_{\text{lvat}}) \tag{18}$$

$$L_{\text{lvat}} := R_{\text{lvat}}(\mathbf{x}, \mathbf{r}_{\text{lvat}}) \tag{19}$$

$$= \text{KL}(f(\mathbf{x}, \theta) \parallel f(\mathbf{x}_{\text{adv}}, \theta)). \tag{20}$$

where $\epsilon_{\text{lvat}}$ is a hyper-parameter to decide the magnitude of $\mathbf{r}_{\text{lvat}}$. The $\epsilon_{\text{vat}}$ in VAT gives the $L_2$ distance $\|\mathbf{x} - \mathbf{x}_{\text{adv}}\|_2$ in the input space, whereas $\epsilon_{\text{lvat}}$ in LVAT

**Algorithm 1** Computation of consistency cost for VAT

**Input:** $X$: random mini-batch from dataset
**Input:** $f()$: classifier being trained
**Input:** $\epsilon_{\mathrm{vat}}$: magnitude of perturbation
**Input:** $\xi$: very small constant, e.g., 1e−6
**Input:** $\mathbf{d}$: random unit vector of same shape of $X$
**Output:** $L_{\mathrm{vat}}$: consistency cost of VAT
1: $\mathbf{g} \leftarrow \nabla_{\mathbf{d}}\ \mathrm{KL}(f(X) \parallel f(X + \xi\mathbf{d})$
2: $\mathbf{r}_{\mathrm{vat}} \leftarrow \epsilon_{\mathrm{vat}}\mathbf{g}/\|\mathbf{g}\|_2$
3: $L_{\mathrm{vat}} \leftarrow \mathrm{KL}(f(X) \parallel f(X + \mathbf{r}_{\mathrm{vat}}))$
4: **return** $L_{\mathrm{vat}}$

**Algorithm 2** Computation of consistency cost for LVAT

**Input:** $X$: random mini-batch from dataset
**Input:** $f()$: classifier being trained
**Input:** Enc() and Dec(): encoder and decoder of transfomer
**Input:** $\epsilon_{\mathrm{lvat}}$: magnitude of perturbation
**Input:** $\xi$: very small constant, e.g., 1e−6
**Input:** $\mathbf{d}$: random unit vector of same size as latent space
**Output:** $L_{\mathrm{lvat}}$: consistency cost of LVAT
1: $Z \leftarrow \mathrm{Enc}(X)$
2: $\mathbf{g} \leftarrow \nabla_{\mathbf{d}}\ \mathrm{KL}(f(X) \parallel f(\mathrm{Dec}(Z + \xi\mathbf{d})))$
3: $\mathbf{r}_{\mathrm{lvat}} \leftarrow \epsilon_{\mathrm{lvat}}\mathbf{g}/\|\mathbf{g}\|_2$
4: $L_{\mathrm{lvat}} \leftarrow \mathrm{KL}(f(X) \parallel f(\mathrm{Dec}(Z + \mathbf{r}_{\mathrm{lvat}})))$
5: **return** $L_{\mathrm{lvat}}$

gives the $L_2$ distance between $\|\mathbf{z} - \mathbf{z}_{\mathrm{adv}}\|_2$ in the latent space. The pseudo-code to obtain $L_{\mathrm{lvat}}$ is shown in Algorithm 2.

The full loss function $L$ is thus given by

$$\begin{aligned} L &= L_{\mathrm{sl}}(D_{\mathrm{l}}, \theta) + \alpha L_{\mathrm{usl}}(D_{\mathrm{l}}, D_{\mathrm{u}}, \theta) \\ L_{\mathrm{usl}} &= \mathbb{E}_{\mathbf{x} \in D_{\mathrm{l}}, D_{\mathrm{u}}}[L_{\mathrm{lvat}}] \end{aligned} \tag{21}$$

where $L_{\mathrm{usl}}$ is the unsupervised loss, i.e., consistency cost, $L_{\mathrm{sl}}$ is a typical supervised loss (cross-entropy for our task), and $\alpha$ is a coefficient relative to the supervised cost.

## 5 Experiments

We evaluate our proposed method in an image classification task using SVHN and CIFAR-10 datasets. Both supervised learning (SL) and semi-supervised learning (SSL) tests are conducted. The experimental code[8] was run with NVIDIA GeForce GTX 1070.

[8] https://github.com/geosada/LVAT

**Table 1.** Architecture of classifier. BNorm stands for batch normalization [12]. Slopes of all Leaky ReLU (lReLU) [24] are set to 0.1.

| | |
|---|---|
| Input: $32 \times 32$ RGB image | 8: $2 \times 2$ max-pool, dropout 0.5 |
| 1: $3 \times 3$ conv. 128 same padding, BNorm, lReLU | 9: $3 \times 3$ conv. 512 valid padding, BNorm, lReLU |
| 2: $3 \times 3$ conv. 128 same padding, BNorm, lReLU | 10: $1 \times 1$ conv. 256 BNorm, lReLU |
| 3: $3 \times 3$ conv. 128 same padding, BNorm, lReLU | 11: $1 \times 1$ conv. 128 BNorm, lReLU |
| 4: $2 \times 2$ max-pool, dropout 0.5 | 12: Global average pool $6 \times 6 \rightarrow 1 \times 1$ |
| 5: $3 \times 3$ conv. 256 same padding, BNorm, lReLU | 13: Fully connected $128 \rightarrow 10$ |
| 6: $3 \times 3$ conv. 256 same padding, BNorm, lReLU | 14: BNorm (only for SVHN) |
| 7: $3 \times 3$ conv. 256 same padding, BNorm, lReLU | 15: Softmax |

### 5.1 Datasets

The street view house numbers (SVHN) dataset consists of $32 \times 32$ pixel RGB images of real-world house numbers, having 10 classes. The CIFAR-10 dataset also consists of $32 \times 32$ pixel RGB images in 10 different classes, *airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships*, and *trucks*. The numbers of training/test images are $73,257/26,032$ for SVHN and $50,000/10,000$ for CIFAR-10, respectively.

We also evaluate our method using augmented datasets. We augmented data using random $2 \times 2$ translation for both datasets and horizontal flips only for CIFAR-10 same as previous study [25]. These augmentations are dynamically applied for each mini-batch. We denote the datasets with data augmentation by (w/ aug.), and our evaluation is conducted with four datasets, SVHN, SVHN (w/ aug.), CIFAR-10, and CIFAR-10 (w/ aug.).

In tests in SL, all labels in the training dataset are used, and the results are averaged over 3 runs. In tests in SSL, $1,000$ and $4,000$ labeled data points are randomly sampled for SVHN and CIFAR-10, respectively. To evaluate different combinations of labeled data in tests in SSL, we prepared 5 different datasets with 5 different seeds for random sampling of labeled data points, and the results are averaged over them.

### 5.2 Model Training

The transformer can be modularized in our method, and thus we first train only the transformer for each dataset separately from the classifier. Once we build the transformers, then, training the classifier with LVAT using the trained transformer model follows. We can use the same transformers throughout all the experiments, which benefits us as it reduces experiment time significantly, especially when we have to run the experiments many times (e.g., for grid-searching for hyper-parameters). This can be viewed as a sort of curriculum strategy that is found for example in [22].

**Model Architectures.** The architecture of the classifier is the same as that of the previous works, and the detail is shown in Table 1. The architecture of the VAE is designed based on DCGAN [29], which is shown in Table 2. The

**Table 2.** Architecture of encoder and decoder of VAE. Dimensionality of latent space is 128. BNorm stands for batch normalization. Slopes of Leaky ReLU (lReLU) are set to 0.1.

| Enc | Dec |
|---|---|
| Input: $32 \times 32 \times 3$ image | Input: 128-dimensional vector |
| $2 \times 2$ conv. 128 valid padding, BNorm, ReLU | Fully connected $128 \rightarrow 512$ ($4 \times 4 \times 32$), lReLU |
| $2 \times 2$ conv. 256 valid padding, BNorm, ReLU | $2 \times 2$ deconv. 512 same padding, BNorm, ReLU |
| $2 \times 2$ conv. 512 valid padding, BNorm, tanh | $2 \times 2$ deconv. 256 same padding, BNorm, ReLU |
| Fully connected $8192 \rightarrow 128$ | $2 \times 2$ deconv. 128 same padding, BNorm, ReLU |
| | $1 \times 1$ conv. 128 valid padding, sigmoid |

architecture of the Glow mainly consists of two parameters: the depth of flow $K$ and the number of levels $L$. We set $K = 22$ and $L = 3$, respectively[9]. Refer to our experimental code for more details.

**Hyper-Parameters.** For the classifier with LVAT, we fixed the coefficient $\alpha = 1$ in Eq. (21), like the original VAT. We used the Adam optimizer [15] with the momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The initial learning rate is set to 0.001 and decays linearly with the last 16,000 updates, and $\beta_1$ is changed to 0.5 when the learning rate starts decaying. The size of a mini-batch is 32 and 128 for $L_{\mathrm{sl}}$ and $L_{\mathrm{usl}}$, respectively for both datasets. We trained each model with 48,000 and 200,000 updates for SVHN and CIFAR-10, respectively. The best hyper-parameter $\epsilon_{\mathrm{lvat}}$ in Eq. (17) was found through a grid search in the SSL setting. For LVAT-VAE, 1.5 and 1.0 were selected for SVHN and CIFAR-10, respectively, from {0.1, 0.25, 0.5, 0.75, 3.0, 4.0, ..., 15.0}. For LVAT-Glow, 1.0 was selected for both SVHN and CIFAR-10 from {0.5, 1.0, 1.5}.

Also for a fair comparison, we conduct the test in SL for VAT, since the results for these were not reported in the original paper except for the one on CIFAR-10 (w/ aug.). According to the code the original authors provide[10], we set $\epsilon_{\mathrm{vat}}$ to $2.5, 3.5, 10.0$, and 8.0 for SVHN, SVHN (w/ aug.), CIFAR-10, and CIFAR-10 (w/ aug.), respectively. Although these values are provided for SSL test and we also attempted other values, it was found that the above ones were better. Regarding $\epsilon_{\mathrm{lvat}}$ and $\epsilon_{\mathrm{vat}}$, we use these values for LVAT and VAT for all experiments unless otherwise noted.

The VAE and Glow were trained using the same data that was used for training the classifier. For the VAE, we also used the Adam optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, with batch size 256. The learning rate starts with 0.001 and exponentially decays with rate 0.97 at every 2 epochs after the first 80 epochs, and we trained for 300 epochs. For the Glow, the learning rate starts with 0.0001, and we trained $3,200$ iterations for SVHN and CIFAR-10 and $5,200$ iterations for SVHN (w/ aug.) and CIFAR-10 (w/ aug.).

[9] We implemented Glow model based on [19].
[10] https://github.com/takerum/vat_tf

**Table 3.** Error rates (%) comparing to VAT and other methods. Results with data augmentation are denoted with (w/ aug.). SSL indicates semi-supervised learning, i.e., number of labeled data $N_l$ is 1,000 and 4,000 for SVHN and CIFAR-10, respectively. SL indicates supervised learning, i.e., all training data are used with label.

| | SVHN | | SVHN (w/ aug.) | | CIFAR-10 | | CIFAR-10 (w/ aug.) | |
|---|---|---|---|---|---|---|---|---|
| Methods | SSL | SL | SSL | SL | SSL | SL | SSL | SL |
| **Consistency Regularization** | | | | | | | | |
| Sajjadi *et al.* [31] | - | - | - | 2.22 (± 0.04) | - | 11.29 (± 0.24) | - | |
| MT [34] | 5.21 (± 0.21) | 2.77 (± 0.09) | 3.95 (± 0.19) | 2.50 (± 0.05) | 17.74 (± 0.30) | 7.21 (± 0.24) | 12.31 (± 0.28) | 5.94 (± 0.14) |
| Π-Model [20] | 5.43 (± 0.25) | - | 4.82 (± 0.17) | 2.54 (± 0.04) | 16.55 (± 0.29) | - | 12.36 (± 0.31) | 5.56 (± 0.10) |
| TempEns [20] | - | - | 4.42 (± 0.16) | 2.74 (± 0.06) | - | - | 12.16 (± 0.24) | 5.60 (± 0.14) |
| VAT [25] | 5.77 (± 0.32) | 2.34 (± 0.05) [1] | 5.42 (± 0.22) | 2.22 (± 0.08) [1] | 16.92 (± 0.45) [2] [3] | 8.175 [2] | 11.36 (± 0.34) | 5.81(± 0.02) |
| **Graph-based Methods** | | | | | | | | |
| LBA [4] [10] | 9.25 (± 0.65) | 3.61 (± 0.10) | 9.25 (± 0.65) | 3.61 (± 0.10) | 19.33 (± 0.51) | 8.46 (± 0.18) | 19.33 (± 0.51) | 8.46 (± 0.18) |
| CCLP [14] | 5.69 (± 0.28) | 3.04 (± 0.05) | - | - | 18.57 (± 0.41) | 8.04 (± 0.18) | - | - |
| **GAN-based Methods** | | | | | | | | |
| ALI [8] | 7.42 (± 0.65) | - | - | - | 17.99 (± 1.62) | - | - | - |
| CatGAN [33] | - | - | - | - | 19.58 (± 0.58) | 9.38 | - | - |
| TripleGAN [21] | 5.77 (± 0.17) | - | - | - | 16.99 (± 0.36) | - | - | - |
| ImprovedGAN [32] | 8.11 (± 1.30) | - | - | - | 18.63 (± 2.32) | - | - | - |
| BadGAN [5] | 4.25 (± 0.03) | - | - | - | 14.41 (± 0.30) | - | - | - |
| LVAT-VAE (Ours) | 4.44 (± 0.36) | **2.26** (± 0.08) | 4.20 (± 0.23) | **2.02** (± 0.04) | 13.90 (± 0.36) | 8.05 (± 0.30) | 14.64 (± 0.54) | 6.54 (± 0.26) |
| LVAT-Glow (Ours) | **4.20** (± 0.45) | **2.23** (± 0.07) | **3.83** (± 0.37) | **2.13** (± 0.07) | **9.94** (± 0.22) | **5.24** (± 0.20) | **7.34** (± 0.24) | **3.94** (± 0.05) |

[1] Results of our experiments with code [25] provided. [2] Results of our experiments with code [25] provided without ZCA.
[3] Reported result in [25] is 14.87 (± 0.38) with ZCA. [4] Results of re-implementation by [14].

**Table 4.** Error rates (%) comparing to combination methods. Notations are same as those in Table 3.

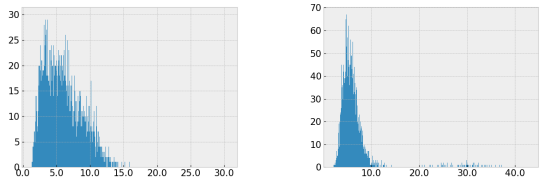
| | SVHN | | SVHN (w/ aug.) | | CIFAR-10 | | CIFAR-10 (w/ aug.) | |
|---|---|---|---|---|---|---|---|---|
| Methods | SSL | SL | SSL | SL | SSL | SL | SSL | SL |
| **Combination Methods** | | | | | | | | |
| MT + SNTG [23] | - | - | 3.86 (± 0.27) | 2.42 (± 0.06) | - | - | - | - |
| MT + fast-SWA [1] | - | - | - | - | - | - | 9.05 (± 0.21) | 4.73 (± 0.18) |
| Π-Model + SNTG [23] | 4.22 (± 0.16) | - | **3.82** (± 0.25) | 2.42 (± 0.05) | 13.62 (± 0.17) | - | 11.00 (± 0.13) | 5.19 (± 0.14) |
| Π-Model + fast-SWA [1] | - | - | - | - | - | - | 10.07 (± 0.27) | 4.72 (± 0.04) |
| TempEns + SNTG [23] | - | - | 3.98 (± 0.21) | 2.44 (± 0.03) | - | - | 10.93 (± 0.14) | 5.20 (± 0.14) |
| VAT + Ent [25] | 4.28 (± 0.10) | - | 3.86 (± 0.11) | - | 13.15 (± 0.21) | - | 10.55 (± 0.05) | - |
| VAT + Ent + SNTG [23] | **4.02** (± 0.20) | - | 3.83 (± 0.22) | - | 12.49 (± 0.36) | - | 9.89 (± 0.34) | - |
| VAT + Ent + fast-SWA [1] | - | - | - | - | - | - | 10.97 | - |
| VAT + LGA [13] | 6.58 (± 0.36) | - | - | - | 12.06 (± 0.19) | - | - | - |
| LVAT-VAE (Ours) | 4.44 (± 0.36) | **2.26** (± 0.08) | 4.20 (± 0.23) | **2.02** (± 0.04) | 13.90 (± 0.36) | 8.05 (± 0.30) | 14.64 (± 0.54) | 6.54 (± 0.26) |
| LVAT-Glow (Ours) | 4.20 (± 0.45) | **2.23** (± 0.07) | 3.83 (± 0.37) | **2.13** (± 0.07) | **9.94** (± 0.22) | **5.24** (± 0.20) | **7.34** (± 0.24) | **3.94** (± 0.05) |

### 5.3 Results

We show classification accuracies. Note that some methods in Tables 3 and 4, including VAT, performed image pre-processing with ZCA on CIFAR-10, which is not used in our experiments. In terms of the model capacity, it is a fair comparison as all other methods used the same network architecture as ours.

In Table 3, we compared LVAT to VAT, other consistency regularizations, and also other approaches introduced in Section 2. We can see that LVAT substantially improved the original VAT, and moreover, outperformed all of other methods in all eight experimental settings.

It has been reported that even better results can be obtained by combining consistency regularizations (MT, Π-Model, TempEns, and VAT) together with other techniques, such as graph-based method (SNTG). We also compared our method to those combinations in Table 4. It is noteworthy that LVAT still sur-




(a) SVHN (b) CIFAR-10

**Fig. 2.** Histograms of $L_2$ distance between the original input images and the adversarial images generated by LVAT-Glow with $\epsilon_{\text{lvat}} = 1.0$. x-axis is $\|\mathbf{x} - \text{Dec}(\text{Enc}(\mathbf{x}) + \mathbf{r}_{\text{lvat}})\|_2$ and y-axis is frequency. For each dataset 5,000 samples are randomly sampled. This indicates that LVAT generates various magnitudes of perturbations.

passed all other methods, except for the result in the SSL testings on SVHN (VAT + Ent + SNTG) and on SVHN (w/ aug.) ($\Pi$-Model + SNTG). In particular, LVAT-Glow on CIFAR-10 and CIFAR-10 (w/ aug.) showed outstanding performance. We believe that combining LVAT with other methods (e.g., LVAT + SNTG) would also achieve further improved performance, but we leave it as future works.

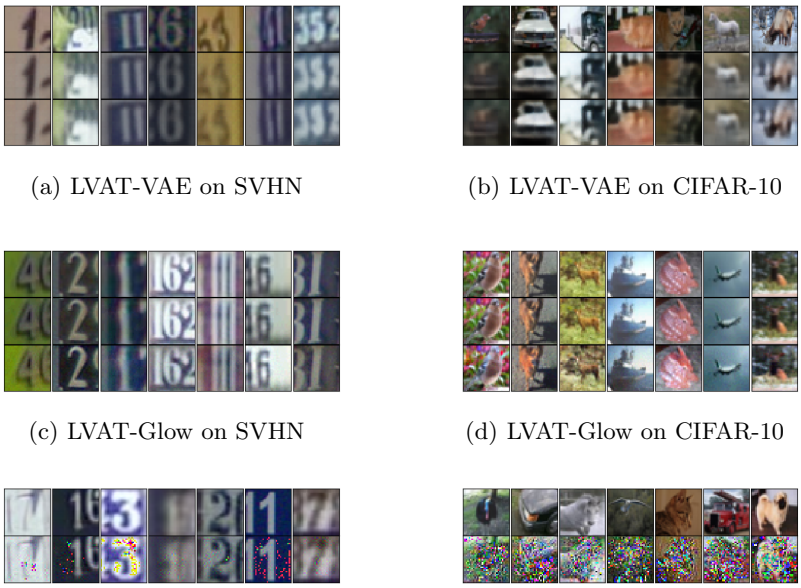
## 6 Discussions

### 6.1 Adversarial Examples

In this section, we analyze our results focusing on the adversarial examples that LVAT generates.

**Perturbation Magnitude.** First, we see the magnitude of perturbation in the input space. Fig. 2 shows the histograms of $L_2$ distance between the original input images and the adversarial images that LVAT generates, i.e., $\|\mathbf{x} - \text{Dec}(\text{Enc}(\mathbf{x}) + \mathbf{r}_{\text{lvat}})\|_2$, which corresponds to $\epsilon_{\text{vat}}$ in the original VAT. It is shown that LVAT generates adversarial examples in the wide range of magnitude, unlike in the original VAT where every adversarial example is generated with the same given magnitude $\epsilon_{\text{vat}}$. In terms of perturbation magnitude, we can see that the LVAT can generate various adversarial examples as we aimed.

**Visual Appearance.** Next, we see the visual appearance of adversarial examples of LVAT and VAT. Fig. 3 shows that adversarial images of VAT are tainted with artifacts, whereas the ones of LVAT look realistic. In both transformer VAE and Glow, the latent space $p(\mathbf{z})$ is constructed so that the points in the high-density area in $p(\mathbf{z})$ correspond to the data used during the model training, i.e., correspond to real images. Thus, unless $\epsilon_{\text{lvat}}$ is not too large, the perturbed latent representation $\mathbf{z}_{\text{adv}}$ computed in LVAT still should correspond



(a) LVAT-VAE on SVHN (b) LVAT-VAE on CIFAR-10

(c) LVAT-Glow on SVHN (d) LVAT-Glow on CIFAR-10

(e) VAT on SVHN (f) VAT on CIFAR-10

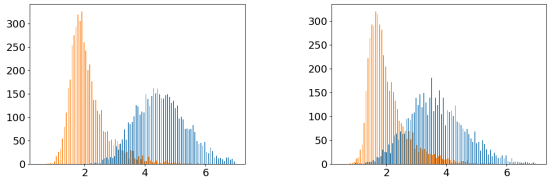
**Fig. 3.** Generated Images. For (a) through (d), first row: original images $\mathbf{x}$, second row: reconstructed images via transformer without perturbation $\hat{\mathbf{x}} = \mathrm{Dec}(\mathrm{Enc}(\mathbf{x}))$, third row: adversarial images $\mathbf{x}_{\mathrm{adv}} = \mathrm{Dec}(\mathrm{Enc}(\mathbf{x}) + \mathbf{r}_{\mathrm{lvat}})$. For (e) and (f), first row: $\mathbf{x}$, second row: $\mathbf{x}_{\mathrm{adv}} = \mathbf{x} + \mathbf{r}_{\mathrm{vat}}$.

to a realistic image. It has been argued in [36] that these noisy images generated in VAT seem harmful to further performance improvement. Unlike VAT, there is no such concern in LVAT.

### 6.2 Failure Analysis: Limitation of VAE Reconstruction Ability on CIFAR-10

Our proposed method LVAT achieved good results as we saw, especially LVAT-Glow on CIFAR-10. However, it also turned out that the error rates of LVAT-VAE on CIFAR-10 were higher than the other experimental settings. We analyze the reason of that in this section.

In Fig. 3(b), we can see that the adversarial images (the third row) on CIFAR-10 are blurred, and more importantly, the images just reconstructed without perturbation (the second row) are also blurry. This indicates that regardless of perturbation, just passing Enc() and Dec() of VAE will blur the input image, which can be viewed as the known VAE characteristics [4]. Fig. 4 shows the reconstruction error in VAE: $L_2$ distance between the original images and the decoded images by VAE for both with and without perturbing, i.e., $\|\mathbf{x} - \mathrm{Dec}(\mathrm{Enc}(\mathbf{x}) + \mathbf{r}_{\mathrm{lvat}})\|_2$ and $\|\mathbf{x} - \mathrm{Dec}(\mathrm{Enc}(\mathbf{x}))\|_2$. It can be seen that the




(a) w/ perturbation (b) wo/ perturbation

**Fig. 4.** Distance from $\mathbf{x}$ ($L_2$ norm) in LVAT-VAE. (a) is $\|\mathbf{x} - \mathrm{Dec}(\mathrm{Enc}(\mathbf{x}) + \mathbf{r}_{\mathrm{lvat}})\|_2$ and (b) is $\|\mathbf{x} - \mathrm{Dec}(\mathrm{Enc}(\mathbf{x}))\|_2$, i.e., reconstruction error. SVHN is orange and CIFAR-10 is blue. For each dataset 5,000 samples are randomly sampled, and y-axis is frequency. This indicates regardless of perturbation, the reconstruction error of VAE is larger on CIFAR-10 than on SVHN.

reconstruction error of VAE is larger on CIFAR-10 than on SVHN, and we think that it is caused by the difference in the complexity of images contained in each dataset.

Contrary to the VAE, the Glow reconstructs very sharp images (the second row in Fig. 3(d)) and the classification performance of LVAT-Glow was very good on CIFAR-10. Given these observations, we conclude that the reconstruction ability of the transformer is crucial to the quality of regularization, which caused the high error rates of LVAT-VAE on CIFAR-10.

## 7 Conclusion

We focused on the local constraint of VAT: VAT can generate adversarial perturbation only within a very small area around the input data point. In order to circumvent this constraint, we proposed LVAT in which computing and injecting perturbation are done in the latent space. Since adversarial examples in LVAT are generated via the latent space, they are more flexible than those in the original VAT, which led to more effective consistency regularization and better classification performance as a result. To the best of our knowledge, this work is the first to introduce the latent space in the context of consistency regularization. We compared LVAT with VAT and other state-of-the-art methods in supervised and semi-supervised scenarios for a classification task in SVHN and CIFAR-10 datasets (both with and without data-augmentation). Our evaluation indicates that LVAT outperforms state-of-the-art methods in terms of classification accuracy in different scenarios.

## Acknowledgement

This work was supported in part by JSPS KAKENHI Grant Number 20K11807.

## References


1. Athiwaratkun, B., Finzi, M., Izmailov, P., Wilson, A.G.: There are many consistent explanations of unlabeled data: Why you should average. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=rkgKBhA5Y7
2. Berthelot, D., Carlini, N., Goodfellow, I., Papernot, N., Oliver, A., Raffel, C.: Mixmatch: A holistic approach to semi-supervised learning. In: NeurIPS (2019)
3. Cao, X., Gong, N.Z.: Mitigating evasion attacks to deep neural networks via region-based classification. In: Proceedings of the 33rd Annual Computer Security Applications Conference. pp. 278–287. ACM (2017)
4. Chen, X., Kingma, D.P., Salimans, T., Duan, Y., Dhariwal, P., Schulman, J., Sutskever, I., Abbeel, P.: Variational lossy autoencoder. In: International Conference on Learning Representations (2017)
5. Dai, Z., Yang, Z., Yang, F., Cohen, W.W., Salakhutdinov, R.R.: Good semi-supervised learning that requires a bad gan. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 6510–6520. Curran Associates, Inc. (2017), http://papers.nips.cc/paper/7229-good-semi-supervised-learning-that-requires-a-bad-gan.pdf
6. Dinh, L., Krueger, D., Bengio, Y.: Nice: Non-linear independent components estimation. In: International Conference on Learning Representations (2015)
7. Dinh, L., Sohl-Dickstein, J., Bengio, S.: Density estimation using real nvp. In: International Conference on Learning Representations (2017)
8. Dumoulin, V., Belghazi, I., Poole, B., Mastropietro, O., Lamb, A., Arjovsky, M., Courville, A.: Adversarially learned inference. In: International Conference on Learning Representations (2017)
9. Fawzi, A., Fawzi, H., Fawzi, O.: Adversarial vulnerability for any classifier. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31, pp. 1178–1187. Curran Associates, Inc. (2018), http://papers.nips.cc/paper/7394-adversarial-vulnerability-for-any-classifier.pdf
10. Haeusser, P., Mordvintsev, A., Cremers, D.: Learning by association – a versatile semi-supervised training method for neural networks. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (July 2017)
11. Huang, C.W., Krueger, D., Lacoste, A., Courville, A.: Neural autoregressive flows. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 2078–2087. PMLR, Stockholmsmssan, Stockholm Sweden (10–15 Jul 2018), http://proceedings.mlr.press/v80/huang18d.html
12. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 37, pp. 448–456. PMLR, Lille, France (07–09 Jul 2015), http://proceedings.mlr.press/v37/ioffe15.html
13. Jackson, J., Schulman, J.: Semi-supervised learning by label gradient alignment. arXiv preprint arXiv:1902.02336 (2019)
14. Kamnitsas, K., Castro, D., Folgoc, L.L., Walker, I., Tanno, R., Rueckert, D., Glocker, B., Criminisi, A., Nori, A.: Semi-supervised learning via compact latent space clustering. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning

Research, vol. 80, pp. 2459–2468. PMLR, Stockholmsmssan, Stockholm Sweden (10–15 Jul 2018), http://proceedings.mlr.press/v80/kamnitsas18a.html
15. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: International Conference on Learning Representations (2015)
16. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. In: International Conference on Learning Representations (2014)
17. Kingma, D.P., Dhariwal, P.: Glow: Generative flow with invertible 1x1 convolutions. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31, pp. 10215–10224. Curran Associates, Inc. (2018), http://papers.nips.cc/paper/8224-glow-generative-flow-with-invertible-1x1-convolutions.pdf
18. Kingma, D.P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., Welling, M.: Improved variational inference with inverse autoregressive flow. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems 29, pp. 4743–4751. Curran Associates, Inc. (2016), http://papers.nips.cc/paper/6581-improved-variational-inference-with-inverse-autoregressive-flow.pdf
19. Kolasinski, K.: An implementation of the GLOW paper and simple normalizing flows lib (2018), https://github.com/kmkolasinski/deep-learning-notes/tree/master/seminars/2018-10-Normalizing-Flows-NICE-RealNVP-GLOW
20. Laine, S., Aila, T.: Temporal ensembling for semi-supervised learning. In: International Conference on Learning Representations (2017)
21. Li, C., Xu, T., Zhu, J., Zhang, B.: Triple generative adversarial nets. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 4088–4098. Curran Associates, Inc. (2017), http://papers.nips.cc/paper/6997-triple-generative-adversarial-nets.pdf
22. Li, Y., Liu, S., Yang, J., Yang, M.H.: Generative face completion. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (July 2017)
23. Luo, Y., Zhu, J., Li, M., Ren, Y., Zhang, B.: Smooth neighbors on teacher graphs for semi-supervised learning. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2018)
24. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: in ICML Workshop on Deep Learning for Audio, Speech and Language Processing (2013)
25. Miyato, T., Maeda, S.i., Koyama, M., Ishii, S.: Virtual adversarial training: a regularization method for supervised and semi-supervised learning. IEEE transactions on pattern analysis and machine intelligence **41**(8), 1979–1993 (2018)
26. Miyato, T., Maeda, S.i., Koyama, M., Nakae, K., Ishii, S.: Distributional smoothing with virtual adversarial training. In: International Conference on Learning Representations (2016)
27. Papamakarios, G., Pavlakou, T., Murray, I.: Masked autoregressive flow for density estimation. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 2338–2347. Curran Associates, Inc. (2017), http://papers.nips.cc/paper/6828-masked-autoregressive-flow-for-density-estimation.pdf
28. Park, S., Park, J., Shin, S.J., Moon, I.C.: Adversarial dropout for supervised and semi-supervised learning. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)

29. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. In: International Conference on Learning Representations (2016)
30. Rezende, D., Mohamed, S.: Variational inference with normalizing flows. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 37, pp. 1530–1538. PMLR, Lille, France (07–09 Jul 2015), http://proceedings.mlr.press/v37/rezende15.html
31. Sajjadi, M., Javanmardi, M., Tasdizen, T.: Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems 29, pp. 1163–1171. Curran Associates, Inc. (2016), http://papers.nips.cc/paper/6333-regularization-with-stochastic-transformations-and-perturbations-for-deep-semi-supervised-learning.pdf
32. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training gans. In: Advances in neural information processing systems. pp. 2234–2242 (2016)
33. Springenberg, J.T.: Unsupervised and semi-supervised learning with categorical generative adversarial networks. In: International Conference on Learning Representations (2016)
34. Tarvainen, A., Valpola, H.: Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 1195–1204. Curran Associates, Inc. (2017), http://papers.nips.cc/paper/6719-mean-teachers-are-better-role-models-weight-averaged-consistency-targets-improve-semi-supervised-deep-learning-results.pdf
35. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of machine learning research **11**(Dec), 3371–3408 (2010)
36. Xie, Q., Dai, Z., Hovy, E., Luong, M.T., Le, Q.V.: Unsupervised data augmentation for consistency training (2019)
37. Zhou, D., Bousquet, O., Lal, T.N., Weston, J., Schölkopf, B.: Learning with local and global consistency. In: Thrun, S., Saul, L.K., Schölkopf, B. (eds.) Advances in Neural Information Processing Systems 16, pp. 321–328. MIT Press (2004), http://papers.nips.cc/paper/2506-learning-with-local-and-global-consistency.pdf