A. Details on the implementation

In this section, we list all the details of the implementation.

Source Code. Our source code is provided in this repository: https://github.com/okyksl/flow-lp.

A.1. Architectures

Generative model (NF) architecture. We use *Glow* [27] for the normalizing flow architecture. For the MNIST [34] and FashionMNIST [63] experiments, we use a conditional, 12-step, Glow-coupling-based architecture similar to [2]. See Table 7 for the details. For the CIFAR-10/100 [30] and SVHN [42] experiments, we use the original Glow architecture described in [27], *i.e.*, 3 scales of 32 steps each containing activation normalization, affine coupling and invertible 1×1 convolution. We adapt an existing PyTorch implementation in [1] to better match the original Tensorflow implementation in [2]. For more details on multi-scale architecture in normalizing flows, see [12].

Generative Model
Input: $\boldsymbol{x} \in \mathbb{R}^{784}, \boldsymbol{y} \in \mathbb{R}^{10}$
GLOWCouplingBlock
PermuteRandom
...
$\times 10$
...
GLOWCouplingBlock
PermuteRandom

GLOWCouplingBlock
Input: $\boldsymbol{x} \in \mathbb{R}^{784}, \boldsymbol{y} \in \mathbb{R}^{10}$
split $\boldsymbol{x} \to \boldsymbol{x}_1, \boldsymbol{x}_2$ $(784 \to 392, 392)$
subnet $\boldsymbol{x}_2 \oplus \boldsymbol{y} \to \mathbf{s}_1, \mathbf{t}_1$ $(402 \to 392, 392)$
affine coupling $\boldsymbol{x}_1, \mathbf{s}_1, \mathbf{t}_1 \to \boldsymbol{z}_1$ $(3\times392 \to 392)$
subnet $\boldsymbol{z}_1 \oplus \boldsymbol{y} \to \mathbf{s}_2, \mathbf{t}_2$ $(402 \to 392, 392)$
affine coupling $\boldsymbol{x}_2, \mathbf{s}_2, \mathbf{t}_2 \to \boldsymbol{x}_2'$ $(3\times392 \to 392)$
concat. $\boldsymbol{z}_1 \oplus \boldsymbol{z}_2$ $(392, 392 \to 784)$

Subnets
Input: $\boldsymbol{x} \in \mathbb{R}^{402}$
linear $(402 \to 512)$
ReLU
linear $(512 \to 784)$
split $(784 \to 392, 392)$

Table 7. Normalizing flow architectures used for our experiments on **MNIST** and **FashionMNIST**. With $c_{in} \to y_{out}$, we denote the number of channels of the input and output of the layer. With $\oplus$, we denote concatenation operation. We use the implementation provided in https://github.com/VLL-HD/FrEIA. For more details on affine coupling layers, see §3.

Classifier architecture. For our experiments on MNIST, we use LeNet-5 [34] with replaced nonlinearity–instead of $\tanh$ we use ReLU, and we initialize the network parameters with truncated normal distribution $\sigma = 0.1$. For the FashionMNIST experiments, we use the same classifier as used in [53]. See Table 8 for more details. For CIFAR-10/100 and SVHN, we use the ResNet-18 [19] architecture as implemented in [9, 66]. This ResNet-18 includes slight modifications over the standard ResNet-18 architecture in order to achieve better performance on CIFAR-10/100. See [3] and [4] for implementation. In particular, the first layer is changed to a 3×3 convolution with stride 1 and padding 1, from the original 7×7 convolution with stride 2 and padding 3. Additionally, the following max-pooling layer is removed. For CIFAR-10, we also use a similarly modified ResNet-20 [18].

A.2. Hyperparameters

Generative Models. For MNIST and FashionMNIST, we use the Adam [26] optimizer with a batch size of 100 and learning rate of 10^{-6} for 100 epochs to train normalizing flows. For CIFAR-10 and SVHN, we use the Adamax [26] optimizer with a learning rate of 0.0005 and weight decay of 0.00005. We use a warmup learning rate schedule for the first 500.000 steps of the training. That is, the learning rate is linearly increased from 0 to the base learning rate 0.0005 in 500.000 steps.

For VAE-GAN training, we run the implementation provided by authors[5] with the default architectures and parameters. That is, for FashionMNIST, we use $\beta = 2.75$, $\gamma = 1$, $\eta = 0$ and latent space size of 10. We use the Adam optimizer with a batch size of 100, learning rate of 0.005, weight decay of 0.0001 and train VAE-GANs for 60 epochs with an exponential decay scheduling of 0.9 for the learning rate. For CIFAR-10, we use the CelebA [35] setup provided (the only 3-channel color dataset provided) and thus use $\beta = 3.0$, latent space size of 25 and 30 epochs instead. Note that we report *On-Learned-Manifold Adversarial Training* from [53] which uses class-specific VAE-GANs. That is, 10 VAE-GAN architectures are trained for both FashionMNIST and CIFAR-10 datasets.

[1] https://github.com/chrischute/glow
[2] https://github.com/openai/glow
[3] https://github.com/facebookresearch/mixup-cifar10
[4] https://github.com/uoguelph-mlrg/Cutout
[5] https://github.com/davidstutz/disentangling-robustness-generalization

LeNet-5
Input: $\boldsymbol{x} \to \mathbb{R}^{1\times28\times28}$
convolution (ker: 5×5, 1 → 6; stride: 1; pad:2)
ReLU
AvgPool2d (ker: 2×2)
convolution (ker: 5×5, 6 → 16; stride: 1; pad:0)
ReLU
AvgPool2d (ker: 2×2)
Flatten (16×5×5 → 400)
linear (400 → 120)
ReLU
linear (120 → 84)
ReLU
linear (120 → 10)
ReLU

CNN from [53]
Input: $\boldsymbol{x} \in \mathbb{R}^{1\times28\times28}$
convolution (ker: 4×4, 1 → 16; stride: 2; pad:1)
Batch Normalization
ReLU
convolution (ker: 4×4, 16 → 32; stride: 2; pad:1)
Batch Normalization
ReLU
convolution (ker: 4×4, 32 → 64; stride: 2; pad:1)
Batch Normalization
ReLU
Flatten (64×3×3 → 576)
linear (576 → 100)
linear (100 → 10)

Table 8. Convolutional Neural Network (CNN) architectures used for our experiments on **MNIST** and **FashionMNIST**. We use *ker* and *pad* to denote *kernel* and *padding* for the convolution layers, respectively. With $h \times w$, we denote the kernel size. With $c_{in} \to y_{out}$, we denote the number of channels of the input and output of the layer.

Discussion on Hyperparameters of Generative Models. As normalizing flows directly optimize the log-likelihood of the data, there are no hyperparameters in their loss function. Additionally, the normalizing flows that we use have a fixed latent dimension equal to the input dimension due to their architectural design. As noted in §5.3, this is in contrast to VAE-GAN used in [53] where the training involves optimizing separate losses for three networks (namely, encoder, decoder, and discriminator) concurrently. Coefficients called β, γ, and η are used to scale reconstruction, decoder, and discriminator loss, respectively. Additionally, the latent size for VAE-GAN is hand-picked for each dataset.

Classifiers. For MNIST, we use the Adam optimizer with a learning rate of 0.001 and weight decay of 0.001. We train LeNet-5 classifiers for 20 epochs with exponential learning decay of rate 0.1 for 10.000 steps. For FashionMNIST, we use the training setup used in [53]. That is, we use the Adam optimizer with a learning rate of 0.01 and weight decay of 0.0001. We train classifiers for 20 epochs with exponential learning decay of rate 0.9 for 500 steps. For CIFAR-10/100, we use the training setup used in [9, 66]. More precisely, we use Stochastic Gradient Descent (SGD) [46] with a batch size of 128, learning rate of 0.1, weight decay of 0.0005, and Nesterov momentum [41] of 0.9. We train ResNet-18 and ResNet-20 classifiers for 200 epochs and multiply the learning rate by 0.2 at epochs $\{60, 120, 160\}$. For SVHN, we use the same optimizer with a weight decay of 0.0001. We train ResNet-18 classifiers for 120 epochs and multiply the learning rate by 0.1 at epochs $\{30, 60, 90\}$.

Data Augmentation. For CIFAR-10/100, we use standard data augmentation akin to [65]. That is, we zero-pad images with 4 pixels on each side, take a random crop of size 32×32, and then mirror the resulting image horizontally with 50% probability. We use such data augmentation for both training the generative and the classifier models. Hence, our normalizing flows are capable of encoding-decoding operations on augmented samples as well. Advanced data augmentation baselines we use in Table 1 [9, 66], also include the same standard data augmentations. However, the VAE-GAN based approach [53] does not use data augmentation in their generative model. To provide a more direct comparison between the performance of two generative models, in §B.2 we conduct an additional study without any data augmentations.

A.3. Metrics

Fréchet Inception Distance. FID [20] aims at comparing the synthetic samples $x \sim p_g$—where p_g denotes the distribution of the samples of the given generative model, with those of the training data of $x \sim p_d$ in a feature space. The samples are embedded using the first several layers of the Inception network. Assuming p_g and p_d are multivariate normal distributions, it then estimates the means $\boldsymbol{m}_g$ and $\boldsymbol{m}_d$ and covariances C_g and C_d, respectively for p_g and p_d in that feature space. Finally, FID is computed as:

$$\mathbb{D}_{\text{FID}}(p_d, p_g) \approx d^2((\boldsymbol{m}_d, C_d), (\boldsymbol{m}_g, C_g)) = \|\boldsymbol{m}_d - \boldsymbol{m}_g\|_2^2 + Tr(C_d + C_g - 2(C_d C_g)^{\frac{1}{2}}), \tag{FID}$$

where d^2 denotes the Fréchet Distance. Note that as this metric is a distance, the lower it is, the better the performance. We used the implementation of FID[6] in PyTorch.

B. Additional Results

B.1. Results on MNIST

Table 9 summarizes our results on MNIST in full data regime. Although the baseline has a very good performance on this dataset, we observe improved generalization.

Perturbation	Train Accuracy	Train Loss	Test Accuracy	Test Loss
Standard	99.80	0.0069	99.24	0.0288
Randomized-LA, ℓ=ℓ_∞, ϵ=0.15	99.78	0.0076	99.28	0.0262
Adversarial-LA, ℓ=ℓ_∞, ϵ=0.05, α=0.01, k=10	99.26	0.0230	99.43	0.0216

Table 9. Train and test accuracy (%) as well as loss on **MNIST**. Comparison with standard training, versus our latent-space perturbations.

B.2. Additional Results on CIFAR-10

Results without Data Augmentation. To provide a direct comparison between two generative models and eliminate the effect of data augmentation, we run additional experiments. Table 10 shows results for our latent perturbations without any data augmentation to train the normalizing flow and the classifier. In line with our FashionMNIST results in §5.3, we observe that both randomized and adversarial latent attacks overperform the standard baseline and the VAE-GAN based approach.

Method	Accuracy
Standard	49.8
VAE-GAN	49.4
Randomized-LA	54.9
Adversarial-LA	58.2

Table 10. Test accuracy (%) on **CIFAR-10**, in the *low-data regime* (5% of training samples) without any data augmentation.

Results with ResNet-20. Table 11 summarizes our results using the ResNet-20, on CIFAR-10. Inline with our ResNet-18 results in §5.1, we observe that both randomized and adversarial latent attacks overperform the standard baseline.

Method	Accuracy	
Standard	65.6	–
Randomized-LA, ℓ=ℓ_2, ϵ=25.	72.7	
Adversarial-LA, ℓ=ℓ_2, $\epsilon = .5$	77.1	

Table 11. Test accuracy (%) on **CIFAR-10** using ResNet-20, in the *low-data regime* (5% of the training set).

Results with Different Attack Parameters. In Table 12, we provide results with varying hyperparameters for the different attacks. Observe that for Adversarial-LA, in the *high* perturbation setting—where $\epsilon = 2.0$, the classifier still didn't fully fit to the training set, but performance in the test set is above the standard baseline.

[6] https://github.com/mseitzer/pytorch-fid

Multi-step Training. We run additional experiments where we sequentially apply different attack hyperparameters in multi-step training with weaker perturbations to increase the performance on the test set. The results are listed in Table 12, denoted with $+$.

Perturbation	**Train Accuracy**	**Train Loss**	**Test Accuracy**	**Test Loss**
Baselines:				
Standard	100.0	0.002	95.2	0.194
PGD, ℓ=ℓ_2, ϵ=2.0, α=0.5, k=10	61.13	0.895	75.7	0.731
PGD, ℓ=ℓ_∞, ϵ=0.03, α=0.008, k=10	77.3	0.521	86.3	0.442
Ours:				
Randomized-LA, ℓ=ℓ_2, ϵ=10.0	99.8	0.007	95.8	0.161
Randomized-LA, ℓ=ℓ_∞, ϵ=0.25	99.5	0.015	**96.3**	0.142
+Randomized-LA, ℓ=ℓ_∞, ϵ=0.15	100.0	0.002	**96.4**	0.133
Adversarial-LA, ℓ=ℓ_2, ϵ=1.0, α=0.5, k=3	99.9	0.005	**96.6**	0.126
Adversarial-LA, ℓ=ℓ_2, ϵ=2.0, α=1.5, k=2	89.1	0.214	95.8	0.134
+Adversarial-LA, ℓ=ℓ_2, ϵ=1.0, α=0.75, k=2	99.2	0.030	96.5	0.114
+Adversarial-LA, ℓ=ℓ_2, ϵ=0.75, α=0.5, k=2	99.7	0.011	**96.7**	0.115
+Randomized-LA, ℓ=ℓ_∞, ϵ=0.25	100.0	0.002	96.5	0.132
+Randomized-LA, ℓ=ℓ_2, ϵ=10.0	100.0	0.002	96.6	0.131

Table 12. Train and test accuracy (%) as well as loss on **CIFAR-10** using ResNet-18. All of the models are trained with the same hyperparameters listed in §A.2. Perturbations listed with the $+$ sign indicates a multi-step training. For example, last row lists the result of the model trained with $P_{adv}^{\ell_2}, \epsilon = 2.0, \alpha = 1.5, k = 2$ for 130 epochs, $P_{rand}^{\ell_\infty}, \epsilon = 0.25$ for 40 epochs and $P_{rand}^{\ell_2}, \epsilon = 10.0$ for 30 epochs. Note that, regardless of multi-step training, the hyperparameters, including the total number of training epochs ($=$ 200), remain fixed across the experiments.