Data-free Knowledge Distillation with Bi-directional Normalizing Flow

Hyunwoo Kim [1]

Abstract

Knowledge distillation, transferring knowledge from a large teacher network to a smaller student, is one of the popular strategies to compress Deep Neural Network. Training student network from teacher network often requires prediction value of original data that was used to train teacher network. However, due to privacy and license issues, it is common to be restricted from the original data. In this work, we propose Bi-Directional Normalizing Flow(BDNF), a novel architecture to generate artificial data from teacher network for data-free knowledge distillation. Our result shows that BDNF generates a realistic representation of images while showing better distillation performance than our baseline approaches. Also, BDNF is applicable for the case we have a small observable dataset, by additionally training BDNF with observable data in reverse direction.

1. Introduction

As a large-size Deep Neural Networks has shown great success, massive computational complexity and storage requirements appeared as a challenge for model deployment. Many techniques have been introduced to compress network in a diverse way(Courbariaux et al., 2015) (Denton et al., 2014)(Han et al., 2015), and Knowledge Distillation (KD) is one of the techniques which is getting attention from the research community. Introduced by (Hinton et al., 2015), Knowledge Distillation is to transfer knowledge from a teacher network to a smaller student model. For KD, a student network is trained with a soft prediction value of teacher model, instead of a hard true label of the data.

For KD, full training data or part of them are needed to attain the prediction values. However, it's sometimes restricted to attain training data used for teacher-network due to privacy and license issues. For instance, in the field of biomedical, ensuring the protection of privacy and the compliance with data protection rules have become central issues for researchers. Thus there are strong needs to distill a teacher network without data used to train the teacher network or alternative dataset.

A possible question here is whether we can generate pseudo-data to train student network, only utilizing information from teacher network. From the fact that the teacher network has seen the input data at least once, we track the distribution of input data which is only focused on the small manifold of input space. The strategy is to design a generator to generate artificial data which cover input conditional data space $p_X(x|y)$ with given information $p_Y(y|x)$ from teacher network. Generator takes noise z and label y as an input, and generate corresponding image. The task of a generator is simplified to map noise space into data space, so that $p_Z(z|y)$ is well matched to $p_X(x|y)$. With the similar concept, (Chen et al., 2019) (Fang et al., 2019), (Ye et al., 2020) proposed designing a generator through adversarial-learning (Goodfellow et al., 2014), and (Yoo et al., 2019) designed generator with noise decoder and diversity loss. Instead of designing a generator, (Yin et al., 2020) prepared training images by synthesizing them through DeepDream.

Taking a new approach, we propose BDNF (Bi-Directional Normalizing Flow), novel architecture to extract training data from the teacher network. BDNF focus on converting noise space $p_Z(z)$ into data space $p_X(x)$ with tractable log-likelihood function. Generator is trained to map noise space $p_Z(z)$ to data space $p_X(x)$ by estimating $p_Z(z|y)$ with $p_X(x|y)$ directly and maximize log-likelihood function.

Also, thanks to the invertible architecture of normalizing flow, it's possible to train BDNF with log-likelihood in both forward and reverse directions. In case we have observable sample data D_X, the desired behavior of generator is to map noise space $p_Z(z)$ into $p_X(x)$ while assuring $p_X(\dot{x})$ is high enough for $\dot{x} \in D_X$. We simply implement this by training normalizing-flow in a reverse way to maximize log-likelihood for D_X.

Our proposed method has novelty in following standpoints,

- BDNF shows a realistic representation of data and good data-free distillation performance.
- BDNF generates data not only for data-free knowledge distillation case but also for the case we have a few observable sample data.
- To the best of our knowledge, BDNF is the first work

to train normalizing-flow in both forward and reverse ways.

2. Background and Related Work

2.1. Data-Free Knowledge Distillation

Original KD methods such as (Hinton et al., 2015), (Park et al., 2019), (Park & Kwak, 2019) train student network with a soft prediction value of teacher network. However, in many cases, training data is not accessible due to privacy concerns. To deal with this problem, some methods have been researched to distill the network without any accessible input data. (Chen et al., 2019) proposed adversarial-learning based approach(Goodfellow et al., 2014), pretrained teacher network is fixed as discriminator while the generator is designed to synthesized images with adversarial training loss. Similarly, (Fang et al., 2019) used both teacher network and student network as a discriminator to fully utilize information in the student network for adversarial learning. (Micaelli & Storkey, 2019) construct network to play min-max game with a new metric, which is basically between teacher and student network.

While above methods take GAN based training approach, (Yoo et al., 2019) takes new approach by directly designing a generator. In contrast to the other model, generator get class label y additionally with noise z as an input. Classifier(teacher) and decoder guess input $\hat{y}$ and $\hat{z}$ with generated image $\hat{x}$, and distance between $\hat{y}$ and y, $\hat{z}$ and z constructs loss to train network. Diversity loss is additionally added for the diversity of synthesized images.

2.2. Normalizing Flow

Introduced by (Dinh et al., 2014), normalizing flow is an effective generative model based on density estimation. For variable $X \in R^D$ and $Z \in R^D$, the purpose of normalizing flow is to transform a well-known distribution $p_Z(z)$ (such as Gaussian) to target distribution $p_X(x)$. Normalizing flow works as mapping function f such that $Z = f(X)$, probability density of x can be estimated through tractable density function $p_Z(z) : R^D \rightarrow R$ through change of variable theorem

$$p_X(x) = p_Z(f(x))|\det(\frac{\partial f(x)}{\partial x})|$$

The assumption here is that we can track the probability density of $p_X(x)$ through the optimized mapping function f. Thus, we train f to be optimized by maximizing log-likelihood for the given observed data $\mathcal{D} = \{x_i\}_{i=1}^N$.

$$\log p_X(\mathcal{D}) = \sum_{i=0}^{N} \log p_Z(f(x)) + \log|\det(\frac{\partial f(x)}{\partial x})|$$

With optimized f, we can simply sample x by sampling z from $p_Z(z)$ and apply inverse function of $f : z = f^{-1}(x)$. Thus, mapping function f should be designed carefully, since it should be tractable to get determinant of Jacobian, $\left|\frac{\partial f(x)}{\partial x}\right|$, and function should be invertible.

There have been many types of research to find a mapping function that satisfies above conditions. Most of the research including (Dinh et al., 2014), (Dinh et al., 2016), (Kingma & Dhariwal, 2018) applies the affine coupling layer, which divides data into partitions and applies affine transformation to each partition sequentially. (Durkan et al., 2019) suggests using spline function instead of affine transformation, showing better representation with less coupling layer. (Grathwohl et al., 2018) applies neural ODE (Chen et al., 2018) instead of affine coupling layer.

Very recently, normalizing flow has been applied for the conditional generative task, which tracks conditional probability of input data $p_X(x|y)$ instead of $p_X(x)$(Ardizzone et al., 2019)(Winkler et al., 2019)(Lugmayr et al., 2020). The label of data or guided image is given as a condition, showing condition-specific image generation.

3. Bi-Directional Normalizing flow

3.1. Data-Free Knowledge Distillation

For data-free knowledge distillation task, we aim to track input data distribution $p_X(x)$ with given $p_Y(y|x)$, pretrained teacher network. We simply assume that with sophisticatedly designed $p_X(x)$, we can transfer knowledge from teacher network to student network. We target to design a mapping function, which converts the noise space $p_Z(z)$ into the input data space $p_X(x)$. Instead of tracking $p_X(x)$ directly, we track conditional probability density $p_X(x|y)$ to utilize information from the teacher network. For random variable X, Y, Z satisfying $X \in R^d, Z \in R^d, Y \in R^c$ and $X \sim p_X(x), Z \sim p_Z(z), Y \sim p_Y(y)$, optimized mapping function f satisfies

$$x = f(z; y, \theta)$$

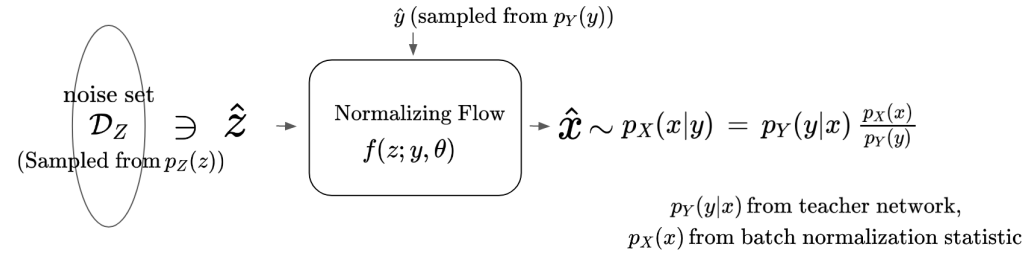
$$p_Z(z|y)dV = p_X(x|y)dV'$$

Change of variable theorem gives

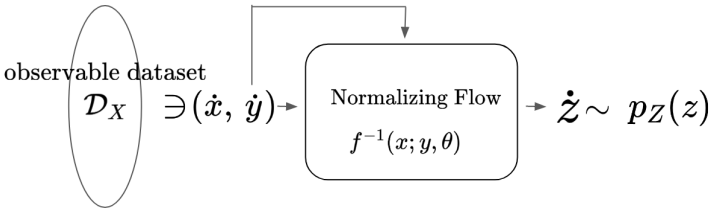
$$p_Z(z|y) = |\det(\frac{\partial f(z;y)}{\partial z})|p_X(x|y)$$

Since $p_X(x|y)$ is not tractable, by applying Bayes's theorem

$$p_Z(z|y) = |\det(\frac{\partial f(z;y)}{\partial z})|\frac{p_Y(y|x)p_X(x)}{p_Y(y)}$$



(a) data-free BDNF training



(b) Bi-directional BDNF training with sample data

Figure 1. Basic concept of BDNF. For data-free KD, normalizing flow is trained to map noise $\hat{z}$ sampled from $p_Z(z)$ to image $\hat{x}$. Generated image $\hat{x}$ follows distribution of $p_X(x)$ which can be converted to $p_Y(y|x)\frac{p_X(x)}{p_Y(y)}$. For BDNF training with sample data, inverse of normalizing flow converts sample data $\dot{x}$ into noise $\dot{z}$, which follows distribution $p_Z(z)$.

For predefined density function $p_Z(z)$, standard Gaussian for example, z and y are independent.

$$p_Z(z) = |\det(\frac{\partial f(z;y)}{\partial z})|\frac{p_Y(y|x)p_X(x)}{p_Y(y)}$$

Unfortunately, we have no way to track $p_X(x)$ directly since we have no way to access input data. Thus, we infer $p_X(x)$ in an indirect way through teacher network, such as statistical information from batch normalization layer(Hinton et al., 2015).

To optimize f, we apply log-likelihood as a criterion. Since it's easy to sample $\hat{z}$ and $\hat{y}$ from predefined distribution $p_z(z)$ and label distribution $p_Y(y)$, we can get log-likelihood of sampled set $\mathcal{D}_Z$, where $(\hat{z}, \hat{y}) \in \mathcal{D}_{\mathcal{Z}}$. Our objective function is as follows.

$$\log p_Z(\mathcal{D}_Z) = \sum_{\hat{z},\hat{y}\in\mathcal{D}_Z} \log|\det(\frac{\partial f(\hat{z};\hat{y})}{\partial \hat{z}})| + \log p_Y(\hat{y}|f(\hat{z};\hat{y})) + \log p_X(f(\hat{z};\hat{y})) - \log p_Y(\hat{y}) \quad (1)$$

where $\mathcal{D}_Z = \{(\hat{z}, \hat{y})|\hat{z}, \hat{y}$ is a number of data each sampled from $p_Z(z)$ and $p_Y(y)\}$

By maximizing log-likelihood, we optimize function f which maps z space into data space. The difference between conventional normalizing flow is that while our method tries to capture input distribution $p_X(x|y)$ by maximizing log-likelihood of $p_Z(z)$, conventional normalizing flow tries to capture input distribution $p_X(x)$ by maximizing log-likelihood of $p_X(x)$. It's possible since we assume we can track $p_X(x|y)$ indirectly by tracking $\frac{p(y|x)p(x)}{p(y)}$ instead.

3.2. Inverse Flow Training With Sample Data

In case we have observable data $\mathcal{D}_X$, training generator additionaly with observable data helps generator to get much realistic images. In probabilistic perspective, the desired behavior of the generator is to capture input data distribution $p_X(x|y)$ from the teacher network while guaranteeing that $p_X(\dot{x}|\dot{y})$ for $\dot{x}, \dot{y} \in \mathcal{D}_X$ is high enough. To accomplish this goal, we suggest multi-task learning of normalizing flow with observable sample data. Thanks to the nature of normalizing flow which is 100% invertible, we can make a model to track $p_X(\dot{x}|\dot{y})$ for $\dot{x}, \dot{y} \in \mathcal{D}_X$. Our strategy is to train normalizing flow with the same way as conventional normalizing flow with sample data, $\mathcal{D}_X$. Thus, objective function can be defined as

$$\log p_X(\mathcal{D}_X) = \sum_{\dot{x},\dot{y}\in\mathcal{D}_X} \log|\det(\frac{\partial f^{-1}(\dot{x};\dot{y})}{\partial \dot{x}})| + \log p_Z(f^{-1}(\dot{x};\dot{y})) \quad (2)$$

where $\mathcal{D}_X = \{\dot{x}, \dot{y}|\dot{x}, \dot{y}$ is sample data from original train

set} and $z = f^{-1}(x; y, \theta)$.

By maximizing log-likelihood for $\mathcal{D}_X$, we can ensure the mapping function f to have a high probability density for $p_X(\dot{x}|\dot{y})$. We propose multi-task learning of normalizing flow, training BDNF with objective functions (1) and (2), to control the behavior of normalizing flow with sample data.

4. Implementation

4.1. Model Structure

We adapt affine coupling layer (Dinh et al., 2014) to design conditional normalizing flow(Ardizzone et al., 2019). Normalizing flow split input data x into $[x_1, x_2]$, and applies affine transformation to each partition alternatively.

$$f_{2i} : x_1 = x_1 \cdot \exp(s_{2i}) + t_{2i},\ x_2 = x_2$$

$$f_{2i+1} : x_1 = x_1,\ x_2 = x_2 \cdot \exp(s_{2i+1}) + t_{2i+1}$$

$$[s_{2i}, t_{2i}] = g_{2i}(x_2; y, \theta),\ [s_{2i+1}, t_{2i+1}] = g_{2i+1}(x_1; y, \theta) \tag{3}$$

Conditional normalizing flow is constructed by the repeated composition of affine coupling function, $f = f_1 \circ f_2 \circ \cdots \circ f_n$. Inverse function of f is simply defined as $f^{-1} = f_n^{-1} \circ f_{n-1}^{-1} \circ \cdots \circ f_1^{-1}$, where

$$f_{2i}^{-1} : x_1 = (x_1 - t_{2i}) \cdot \exp(-s_{2i}),\ x_2 = x_2$$

$$f_{2i+1}^{-1} : x_1 = x_1,\ x_2 = (x_2 - t_{2i+1}) \cdot \exp(-s_{2i+1})$$

4.2. Objective Function Implementation

Data-Free Case. Our strategy to implement objective function (1) is in following way.

$$criterion_{distill} = \sum_{\hat{z},\hat{y} \in \mathcal{B}_Z} \Big(CLS(\hat{z}, \hat{y}) + \alpha Jacobian(\hat{z}, \hat{y}) \Big) + \beta L_{bn} \tag{4}$$

Three terms $CLS(\hat{z}, \hat{y})$, $Jacobian(\hat{z}, \hat{y})$ and $\mathcal{L}_{bn}$ consist of training criterion, to mimic log-likelihood. $\mathcal{B}_Z$ is a batch of noise dataset $\mathcal{D}_Z$. α and β are hyperparameters to control the the balance of each criterion term. $CLS(z, y)$ works as $\log p(y|x)$ term in (1).

For conditional normalizing flow f and classifier C, $CLS(z, y)$ is defined as inner product of a sampled label y and log output of C for a artificial image.

$$CLS(z, y) = y \cdot \log C(f(z; y)) \tag{5}$$

$Jacobian(z, y)$ works as $\log |\det(\frac{\partial f(z;y)}{\partial z})|$ term in (1). For normalizing flow with affine coupling layer, log determinant of Jacobian is simply calculated by summing up scaling factor of each coupling layer(Dinh et al., 2014). For the number of coupling layer n and dimension of partitioned data d, $Jacobian(z, y)$ is defined as

$$Jacobian(z, y) = \sum_{i=1}^{n} \sum_{k=1}^{d} s_{ik} \tag{6}$$

for s_i defined in (3).

In order to track $p_X(x)$ terms in (1) without observable data, we utilize statistic information of batch normalization layer in C(Yin et al., 2020). Batch normalization layer contains the mean and variance of features for training data in each layer. Although the actual feature values of each layer follows a much complicate distribution, assuming it as Gaussian brings us an approximated value of $p_X(x)$. For feature value of artificial images to follow Gaussian statistic of batch normalization layer, we added loss term $\mathcal{L}_{bn}$

$$\mathcal{L}_{bn} = -\sum_{l} \|\mu_l(f(z; y)) - \mathbb{E}(\mu_l(x)|\mathcal{X})\|_2 + \|\sigma_l^2(f(z; y)) - \mathbb{E}(\sigma_l^2(x)|\mathcal{X})\|_2 \tag{7}$$

for $\mathcal{X}$ is training data of C and μ_l and $sigma_l^2$ denotes batch-wise mean and variance estimates of l^{th} convolutional layer.

Bi-Directional Training With Observable Sample Data. For the case we have observable sample data, we can train normalizing flow to capture sample data with high probability. With the objective function (2), the training criterion is simply defined as followed.

$$criterion_{rev} = \sum_{\dot{x},\dot{y} \in \mathcal{B}_X} \log p_Z(f^{-1}(\dot{x}; \dot{y})) - Jacobian(\dot{x}, \dot{y}) \tag{8}$$

For the multi-task learning of normalizing flow in both forward and reverse direction, we combine loss functions (4) and (8),

$$criterion_{multi} = criterion_{distill} + \gamma criterion_{rev} \tag{9}$$

γ is a hyperparameter that control balance of multi-task learning. Too high γ makes generator to generate data only confined to sample data, too low γ makes generator hard to catch information from sample data. Thus it's important to find proper value for γ.

5. Experiment

5.1. Proof of Concept With Toy Data

For the proof of concept, we checked BDNF for the 2D toy dataset. We manually generate simple 2D data which is

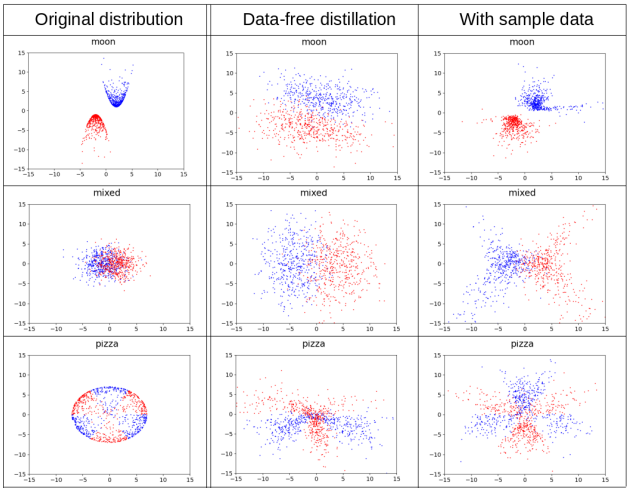
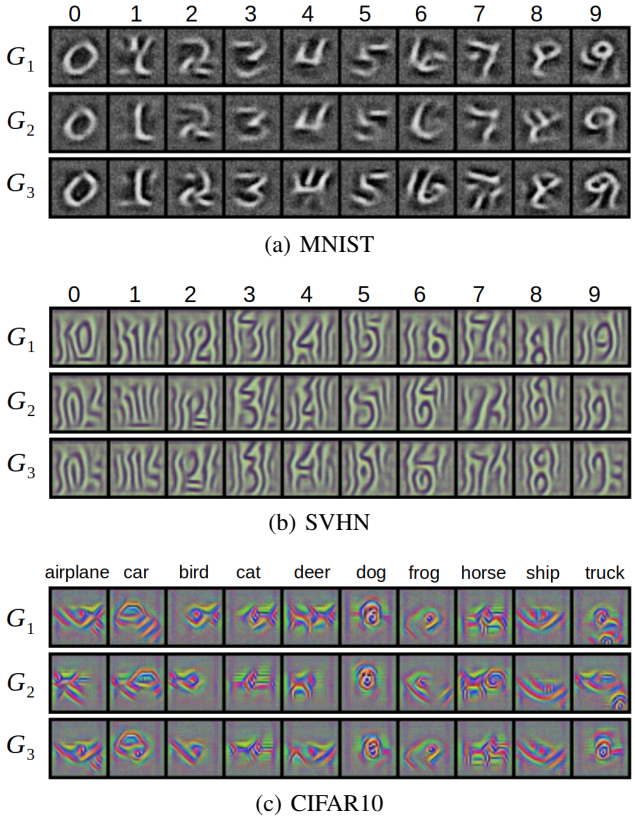


Figure 2. 2D data distributions generated by BDNF. For the toy data labeled with two classes (left), the teacher network is trained to classify input data. BDNF generate artificial data from the teacher network without any observable data (middle). Artificial data from BDNF shows closer distribution to the original distribution with the help of observable sample data(right). Sample data consists of 10 samples for each class, which are randomly sampled from the original distribution.



(a) MNIST

(b) SVHN

(c) CIFAR10

Figure 3. Artificial images generated by BDNF from teacher network trained with MNIST, SVHN and CIFAR10. No observable data was used to generate images.

classified into two classes. We designed the teacher network with multi-layer perceptron(MLP) with 2 hidden layers and trained teacher to classify the toy dataset. Then, BDNF is trained with this teacher network to generate artificial data with given $\hat{z}$ and $\hat{y}$, each sampled from Gaussian $\mathcal{N}(0, 1)$ and discrete uniform distribution of each class. Additionally, for the case we have observable sample data, we construct sample data by randomly selecting 10 samples from each class. We also train BDNF by multi-task learning with the criterion (9). For the entire process, we omit loss term from batch normalization statistic since our MLP teacher network don't include batch normalization layer. Figure 2 visualizes artificial data from BDNF for both data-free and with sample data cases. BDNF generates artificial data representing boundary lines from the teacher network, $p_Y(y|x)$. One thing to note here is that BDNF generates much similar distribution to the original distribution even with small sample data.

5.2. Distillation With Image Dataset

For validation of our design choices, we show the result of data-free knowledge distillation with image dataset.

Baseline Setting. To compare BDNF with prior works, we select KegNegt(Yoo et al., 2019) as our baseline. In the work of KegNet, the author synthesizes data using generator, and trains student network which is compressed from teacher network by Tucker-2 decomposition(Rabanser et al., 2017). We selected KegNet as our baseline since both models have a lot of similarities for the following reasons. First, compared to the other GAN based data-free KD approach, both KegNet and BDNF don't use information from the student network to train generator, which means model performance is independent from the choice of student network. Second, both models take sampled label $\hat{y}$ as an input of generator.

The author of KegNet opened their source code and exact experiment setting to train KegNet on GitHub(Yoo, 2020). We reimplement the experiment with the same setting as the author published. In addition to KegNet, we compare the performance of training student network with data from a random distribution. We use the Gaussian distribution $\mathcal{N}(0, 1)$ and uniform distribution $\mathcal{U}(-1, 1)$ as our additional baseline.

Dataset. We evaluate BDNF on three dataset, MNIST(LeCun et al., 1998), Fashion MNIST(Xiao et al., 2017), and SVHN(Netzer et al., 2011). For the teacher network, we use LeNet5(LeCun et al., 1995) for MNIST, and ResNet14(He et al., 2016) for Fashion MNIST and SVHN.

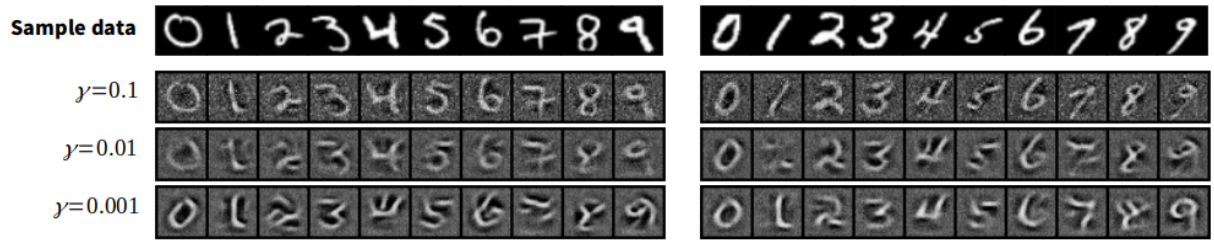


Figure 4. MNIST images from BDNF trained bi-directionally with sample data. We find BDNF shows ideal multi-task learning when $\gamma = 0.01$ since images from $\gamma = 0.1$ are too confined to sample data, and $\gamma = 0.001$ can't capture features of sample data enough.

Implementation Details. We borrowed the experiment setting for from (Yoo et al., 2019). We trained teacher network from scratch and compressed the teacher network by Tucker-2 decomposition of weight matrix for convolution layer while fixing fully-connected layer. We compared 3 student networks with different compression ratios for the generalized comparison. For LeNet5, we compressed the last convolution layer for student 1 and the last two convolution layer for student 2. We increased the compression ratio for each layer for student 3. For ResNet14, we compressed the last two residual blocks and controlled the compression ratio for each of the 3 student networks.

One thing to note is that accuracy from (Yoo et al., 2019) is not reproducible. Since the result from data distillation is dependent on parameter distribution of teacher network, we retrained teacher network and compared KegNet and BDNF from same network for the fair comparison. Additional difference between setting from KegNet and our setting is that while batch normalization layer was not fixed for training in KegNet setting, we found that fixing batch normalization layer for training student network gives us better performance generally. Thus we fix batch normalization layer for ResNet14 after tucker decomposition. Also, while KegNet evaluated KD performance of 5 different generators combined, we compare the performance of only one generator for a fair comparison.

We additionally evaluate the performance of BDNF for the case we have small observable data. We simply construct a sample dataset by randomly picking 1 image from each class. Thus, a total of 10 images consists of a sample dataset for each MNIST, Fashion MNIST, and SVHN.

For training BDNF, we use Adam optimizer(Kingma & Ba, 2014) with a learning rate 10^{-4}. For criterion (4), α is set as 10^{-2}, β is set as 1. For multi-task learning with sample dataset, we trained model with criterion (9) with $\gamma = 10^{-2}$.

For training a student network, we use Adam optimizer with a learning rate 10^{-5}. We simply construct loss function as Kullback-Leibler(KL) divergence of each of prediction value from student and teacher network. Each of the student networks is all trained with the same hyperparameters for the fair comparison.

Artificial Image Generation. Figure 3 shows artificial images of BDNF from teacher network trained with MNIST, SVHN, and CIFAR10. 3 different generators were trained from the same teacher network. We fix noise variable z as zero vector for visualization.

Figure 5.2 shows artificial images generated from BDNF bi-directionally trained with 10 sample images. While BDNF tends to generate images from the teacher network, generated images capture features from sample images.

Evaluation Result. Table 5.2 shows the classification accuracy of BDNF and baseline approaches. In MNIST, training student network with BDNF outperformed KegNet, although KegNet shows better performance than random(Gaussian, Uniform) distribution. Also, BDNF outperforms baseline approach for SVHN and Fashion MNIST dataset, while KegNet shows no better performance than random distribution. One thing to note is that while (Yoo et al., 2019) failed to train student network by not fixing batch normalization layer after tucker decomposition, we fix batch normalization during training process. We successfully train the student network even with uniform and normal distribution, which show fairly good performance as KegNet.

Bi-directional training of BDNF with sample data generally shows better performance than data-free BDNF. Training student networks only with sample dataset give far less performance than bi-directionally trained BDNF with a sample dataset.

6. Conclusion

In this paper, we propose BDNF, a novel approach for data-free KD. We train conditional normalizing flow to have maximum likelihood of $p_Z(z)$, by converting $p_X(x|y)$ to $p_Y(y|x)\frac{p_X(x)}{p_Y(y)}$. BDNF is applicable not only for the case

Table 1. Classification accuracy of BDNF and the baselines on the image datasets. We use three variants of students with different compression ratios for each dataset. Accuracy of BDNF bi-directionally trained with sample data is also evaluated. Randomly selected 10 images from the image dataset construct sample images.

Dataset	Model	Approach	Student1	Student2	Student3
MNIST	LeNet5	original	98.9%	98.9%	98.9%
MNIST	LeNet5	Tucker(T)	24.3%	28.0%	16.2%
MNIST	LeNet5	T+Uniform	64.9%	56.4%	53.4%
MNIST	LeNet5	T+Normal	62.6%	57.5%	49.8%
MNIST	LeNet5	T+KegNet	76.5%	76.2%	61.6%
MNIST	LeNet5	**T+BDNF**	**86.1%**	**83.6%**	**74.3%**
MNIST	LeNet5	T+Sample	68.73%	67.8%	41.8%
MNIST	LeNet5	**T+BDNF(Sample)**	**89.1%**	**82.4%**	**79.5%**
SVHN	ResNet14	original	93.2%	93.2%	93.2%
SVHN	ResNet14	Tucker(T)	13.1%	26.9%	19.8%
SVHN	ResNet14	T+Uniform	69.5%	73.8%	46.8%
SVHN	ResNet14	T+Normal	70.57%	63.8%	22.9%
SVHN	ResNet14	T+KegNet	51.0%	65.1%	30.3%
SVHN	ResNet14	**T+BDNF**	**72.1%**	**78.0%**	**66.5%**
SVHN	ResNet14	T+Sample	60.7%	46.5%	28.1%
SVHN	ResNet14	**T+BDNF(Sample)**	**82.8%**	**79.8%**	**72.3%**
Fashion	ResNet14	original	91.2%	91.2%	91.2%
Fashion	ResNet14	Tucker(T)	71.5%	62.5%	43.1%
Fashion	ResNet14	T+Uniform	$<$71.5%	78.5%	50.4%
Fashion	ResNet14	T+Normal	79.1%	72.3%	58.1%
Fashion	ResNet14	T+KegNet	$<$71.5%	73.7%	59.6%
Fashion	ResNet14	T+BDNF	**81.1%**	**79.0%**	**62.9%**
Fashion	ResNet14	T+Sample	$<$71.5%	68.1%	55.2%
Fashion	ResNet14	T+BDNF(Sample)	**82.8%**	**81.0%**	**62.9%**

no data is observable but also for the case we have a few observable data. BDNF shows realistic images representation with no observable data and generates images similar to given observable sample data. For KD, BDNF shows better performance than our baseline approaches. We checked training BDNF bi-directionally with sample data generally gives better performance than data-free BDNF.

References

Ardizzone, L., Lüth, C., Kruse, J., Rother, C., and Köthe, U. Guided image generation with conditional invertible neural networks. *arXiv preprint arXiv:1907.02392*, 2019.

Chen, H., Wang, Y., Xu, C., Yang, Z., Liu, C., Shi, B., Xu, C., Xu, C., and Tian, Q. Data-free learning of student networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3514–3522, 2019.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in neural information processing systems*, pp. 6571–6583, 2018.

Courbariaux, M., Bengio, Y., and David, J.-P. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28:3123–3131, 2015.

Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. *Advances in neural information processing systems*, 27:1269–1277, 2014.

Dinh, L., Krueger, D., and Bengio, Y. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. Neural spline flows. In *Advances in Neural Information Processing Systems*, pp. 7511–7522, 2019.

Fang, G., Song, J., Shen, C., Wang, X., Chen, D., and Song, M. Data-free adversarial distillation. *arXiv preprint arXiv:1912.11006*, 2019.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. *Advances in neural information processing systems*, 27:2672–2680, 2014.

Grathwohl, W., Chen, R. T., Bettencourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28: 1135–1143, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, D. P. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in neural information processing systems*, pp. 10215–10224, 2018.

LeCun, Y., Jackel, L. D., Bottou, L., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Muller, U. A., Sackinger, E., Simard, P., et al. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, 261:276, 1995.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lugmayr, A., Danelljan, M., Van Gool, L., and Timofte, R. Srflow: Learning the super-resolution space with normalizing flow. In *European Conference on Computer Vision*, pp. 715–732. Springer, 2020.

Micaelli, P. and Storkey, A. J. Zero-shot knowledge transfer via adversarial belief matching. In *Advances in Neural Information Processing Systems*, pp. 9551–9561, 2019.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. 2011.

Park, S. and Kwak, N. Feed: Feature-level ensemble for knowledge distillation. *arXiv preprint arXiv:1909.10754*, 2019.

Park, W., Kim, D., Lu, Y., and Cho, M. Relational knowledge distillation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3967–3976, 2019.

Rabanser, S., Shchur, O., and Günnemann, S. Introduction to tensor decompositions and their applications in machine learning. *arXiv preprint arXiv:1711.10781*, 2017.

Winkler, C., Worrall, D., Hoogeboom, E., and Welling, M. Learning likelihoods with conditional normalizing flows. *arXiv preprint arXiv:1912.00042*, 2019.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Ye, J., Ji, Y., Wang, X., Gao, X., and Song, M. Data-free knowledge amalgamation via group-stack dual-gan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12516–12525, 2020.

Yin, H., Molchanov, P., Alvarez, J. M., Li, Z., Mallya, A., Hoiem, D., Jha, N. K., and Kautz, J. Dreaming to distill: Data-free knowledge transfer via deepinversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8715–8724, 2020.

Yoo, J. Kegnet. https://github.com/snudatalab/KegNet, 2020.

Yoo, J., Cho, M., Kim, T., and Kang, U. Knowledge extraction with no observable data. *Advances in Neural Information Processing Systems*, 32:2705–2714, 2019.