Distilling Normalizing Flows

Steven Walton[1,2,4*] Valeriy Klyukin[3] Maksim Artemev[4]
Denis Derkach[3] Nikita Orlov[4] Humphrey Shi[2,4]
[1]University of Oregon [2]SHI Lab @ GaTech [3]HSE University [4]Picsart AI Research

Abstract

Explicit density learners are becoming an increasingly popular technique for generative models because of their ability to better model probability distributions. They have advantages over Generative Adversarial Networks due to their ability to perform density estimation and having exact latent-variable inference. This has many advantages, including: being able to simply interpolate, calculate sample likelihood, and analyze the probability distribution. The downside of these models is that they are often more difficult to train and have lower sampling quality.

Normalizing flows are explicit density models, that use composable bijective functions to turn an intractable probability function into a tractable one. In this work, we present novel knowledge distillation techniques to increase sampling quality and density estimation of smaller student normalizing flows. We seek to study the capacity of knowledge distillation in Compositional Normalizing Flows to understand the benefits and weaknesses provided by these architectures. Normalizing flows have unique properties that allow for a non-traditional forms of knowledge transfer, where we can transfer that knowledge within intermediate layers. We find that through this distillation, we can make students significantly smaller while making substantial performance gains over a non-distilled student. With smaller models there is a proportionally increased throughput as this is dependent upon the number of bijectors, and thus parameters, in the network.

1. Introduction

Generative models are a popular form of deep learning, due to their ability to generate convincing samples from a probability distribution. The de facto choice for image sampling are Generative Adversarial Networks (GANs) [7]. While GANs have been shown to produce high-quality samples that look similar to the samples from target distribution, they do not faithfully reproduce the target probability distribution [23]. Arora *et al*. [1] demonstrate this by looking at the Birthday Paradox and analyzing the diversity of images. Anyone who has used popular "this x does not exist" websites will quickly notice how similar some images are to those they were trained on. This is due to that GAN's non-exact latent-variable inference, requiring them to have a smaller latent representation and thus being implicit density learners. On the other hand, Normalizing Flows have full latent representations, not using any encoders or decoders as are used in GANs or VAEs, and allow them to learn tractable density functions. Not only may researchers and practitioners want to produce high quality samples, but some may also want to perform density estimation and analyze the target distribution. Explicit density learners, such as Normalizing Flows, are able to accomplish all of these tasks. The major challenge is that it is substantially more challenging to learn all this extra information and require larger latent representations. Variational autoencoders [29] try to solve this by attempting to learn only a subset of the latent space, reducing the dimensionality of the problem to a set of approximated principle components. On the other hand, there are autoregressive models (AR) [6, 34], normalizing flows (NFs) [29], and deep energy-based models (EBMs) [17, 31], which attempt to learn all of the high dimensional data and likelihood information. While these models reproduce the target distribution more faithfully, they are much more difficult to train and require significantly more computational resources.

The advantage of NFs over other explicit density models is that they are simple in structure and fully tractable throughout the entire model. They accomplish their learning by composing bijective functions (equations whose maps are one-to-one and onto). This means that NFs perform a change of variables operation, changing an intractable probability distribution into a tractable one, typically Gaussian. NFs do this through the use of a learned change of variables mapping. The advantage of this is that once we have found the proper simpler density function, we can easily analyze it and gain all the statistical advantages of a tractable probability density function.

Normalizing Flows are particularly difficult to train due to their invertible nature, requiring *diffeomorphic* compositions and thus we require a tractable Jacobian determinants

to preform inference through the backwards pass. While tractable Jacobian determinants are not required for forward inference, it is the bidirectional nature that makes NFs particularly unique. This aspect has been the major challenge of NFs, as it is difficult to find bijective functions that are both efficient and highly expressive [15, 26]. Current techniques lead to a high number of parameters, meaning that they also tend to be inefficient during inference. This work focuses on using knowledge distillation [9] to train smaller models through a student-teacher paradigm. While smaller models are more difficult to train to high accuracy, they perform faster inference due to the reduction in the number of parameters. NFs have unique properties that allow for unique types of knowledge distillation techniques compared to other more traditional models. Specifically, since NFs focus on a change of variables, they learn full latent representations through intermediate layers accurately, which can be distilled between models of similar architectures.

The study of distillation within Normalizing Flow architectures is understudied. In this paper, we explore this unique property and demonstrate its improvements in student models. In this manner we hope to provide a general framework in which distillation may be applied to these networks. We do not seek to find optimal settings or methods of distillation, but to study the capacity of these models to distill knowledge. More specifically, to do so by taking advantage of the unique properties of these architectures and not found within other architectures, such as GANs or VAEs. Furthermore, we believe the insights gained from this work may inspire research for architectures such as diffusion models [10, 32] which share some, but not all, properties.

Our main contributions in this paper are:

- We propose novel knowledge distillation techniques for normalizing flows, demonstrating their effectiveness on smaller flow models.
- We demonstrate the effects of different distillation techniques on these normalizing flows.
- We demonstrate that these methods can be used on various datasets, including tabular datasets and common image datasets.
- We provide a foundation for studying distillation methods applied to Normalizing Flows.

2. Related Work

2.1. Normalizing Flows

Normalizing Flows are a type of explicit density model, where instead of attempting to just generate samples similar to our target distribution, the objective is to model the entire distribution. This can be accomplished through a change of variables method, where a transformation f is used that is both invertible, and where f and f^{-1} are both differentiable, meaning that f is *diffeomorphic*. This change of variables can be expressed as

$$p_x(\boldsymbol{x}) = p_u(\boldsymbol{u}) \left|\det J_f(\boldsymbol{u})\right|^{-1} \tag{1}$$

where $\boldsymbol{u} = f^{-1}(\boldsymbol{x}) = g(\boldsymbol{x})$. p_x represents the distribution being modeled, and p_u represents the learned distribution. In other words, this change of variables is expressed as a learned distribution times the absolute value of the inverse of the Jacobin determinant, det$J$. This inverse Jacobian determinant is why Normalizing Flows are computationally expensive.

Because these functions are diffeomorphic, they are also composable. Therefore, with an arbitrary transformation f_i, a normalizing flow can be created, which is expressed as

$$\boldsymbol{f} = \boldsymbol{f}_1 \circ \boldsymbol{f}_2 \circ \cdots \circ \boldsymbol{f}_k, \tag{2}$$

We refer to the number of composable functions as the depth of the flow network, k. The Jacobian determinant of the entire network can be calculated by the product of the Jacobian determinant at each layer.

$$\det J_{\boldsymbol{f}}(\boldsymbol{x}) = \prod_{i=1}^{n} \det J_{\boldsymbol{f}_i}(\boldsymbol{x}_i) \tag{3}$$

Triangular decomposition is frequently used to solve the Jacobian determinant, as the determinant of a triangular matrix is trivial to compute.

With this, any sample x, from the intractable distribution, p_x, can be fed into the model (function f), and transformed to a tractable distribution p_u. Then, from the tractable distribution, p_u, new samples can be generated by inverting the flow. We refer to the inverse, f^{-1}, as g to denote that this direction is a generator, where one can sample from a distribution similar to p_x.

There are several types of normalizing flow architectures: Planar and Radial Flows introduced by Rezende and Mohamed [29]; Coupling Flows, proposed by Dinh et al. [5]; Autoregressive Flows, presented by Kingma et al. [14]; Continuous Flows, suggested by Chen et al. [3].

Each of these has different trade-offs between expressiveness and simplicity to calculate, with Planar and Radial Flows being the most computationally complex.

Since Coupling and Autoregressive Flows are the most common, we focus on the most popular representatives of each: GLOW (Kingma and Dhariwal [13]) and Masked Autoregressive Flow (Papamakarios et al. [25]).

Normalizing flows are also included in the class of likelihood-based learners, or neural networks that attempt to solve the maximum likelihood of the distribution. Because of this, one can use any metric to minimize the discrepancy

between the target distribution and the learned one. Typically the Kullback-Leibler (KL) divergence is used, which uses entropy between the two distributions to determine the difference. Therefore, the loss for a sample $\boldsymbol{x}$ and parameter $\boldsymbol{\theta}$ can be expressed as:

$$\mathcal{L}(\theta) = D_{KL}\left[p_x(\boldsymbol{x})||p_u(\boldsymbol{x};\boldsymbol{\theta})\right] \tag{4}$$

$$= \sum p_x(x) \log\left(\frac{p_x(x)}{p_u(x)}\right) \tag{5}$$

While this might not always be possible, there are many alternative equivalents such as the reverse KL-Divergence and Evidence Lower Bound (ELBO), which can be used in cases where sampling from the target distribution may not be possible.

Additionally, when training on discrete variables, such as images or text, the dataset typically needs to be dequantized to transform them into continuous distributions. While there are methods to work on discretized data [11, 19], like images, the common practice is to first dequantize the data. For images, the image values can be converted from integers in $[0, 255]$ to real numbers in $[0, 1]$, noise is added to continuize (dequantize) the data and transform it into a tractable density, typically Gaussian. This also means that the dequantization process is, approximately, invertible, which makes this equivalent to another flow step.

2.1.1. GLOW

GLOW is an extension of the RealNVP [5] architecture, which uses affine coupling layers. Among the innovations proposed in the original paper were the activation normalization layer (ActNorm) and the 1×1 invertible convolution layer. ActNorm has the following form:

$$\forall i, j : \boldsymbol{x}_{i,j} = \boldsymbol{s} \odot \boldsymbol{z}_{i,j} + \boldsymbol{b} \tag{6}$$

$\boldsymbol{b}$ and $\boldsymbol{s}$ are initialized so that the mean and standard deviation, respectively, of the first mini-batch of data is 0 and 1. An invertible 1×1 convolution is arranged as follows:

$$\forall i, j : \boldsymbol{x}_{i,j} = W\boldsymbol{z}_{i,j} \tag{7}$$

where $W \in \mathbb{R}^{c\times c}$ is initialized as a square orthonormal matrix of the same size as the number of channels, c. With this setup we can more easily calculate the Jacobian determinant through LU-decomposition. We can further decompose U by giving it a unitary diagonal and adding it to a diagonal matrix where entries are that of the original U matrix:

$$W = PL(U + diag(\boldsymbol{s})), \tag{8}$$

where P is a permutation matrix, L is a lower triangular matrix with unit diagonal, U is an upper triangular matrix.

By pulling the trace of U out, this allows for a simplified calculation of the log Jacobian determinant, since the determinant of a diagonal matrix is the sum of its trace.

$$\log(\det(J)) = h \cdot w \cdot \sum \log|s| \tag{9}$$

where h and w are the height and width (number of pixels) of the input image. We note that while LU-Decomposition significantly reduces the computational complexity, that this function is still rather costly to calculate.

Coupling layers may be expressed in several forms, often additive or affine. These layers are highly expressive flows that operate on a disjoint partition.

2.1.2. Masked Autoregressive Flow

(MAF), models each element of the vector $\boldsymbol{x}$ as a conditioned function $h(\cdot, \theta)$:

$$\boldsymbol{x}_i = h(\boldsymbol{z}_i, \Theta_i(\boldsymbol{z}_{1:i-1})) \tag{10}$$

The determinant of the Jacobian for such a transformation is a lower triangular matrix since $\boldsymbol{x}_i$ depends only on $\boldsymbol{z}_{1:i}$, thus making it trivial to calculate. The calculation of $\boldsymbol{x}$ can be performed with a single network pass by masking the Θ and h network layers.

2.2. Knowledge Transfer and Distillation

With knowledge transfer one can incorporate additional loss information into a model being trained by using output from an already pretrained model. Typically we do this from a model that is larger, called a teacher, that will *distill* the information to a smaller model, called a student. This can help the student model both learn faster and avoid local minima that it might typically get trapped in. There are several schemes of KD:

- Response-based KD: constraining student to produce outputs similar to that of the teacher's. This method is the standard for KD of discriminative models and was first proposed by Hinton et al. [9].
- Feature-based KD: constraining student to have internal representations similar to the teacher's. Similar method was proposed by Romero et al. [30] for general knowledge distillation framework.
- Relation-based KD: training a student to replicate teacher's relationships between different layers or different data points.

A recent work by Baranchuk *et al*. [2] proposed using knowledge distillation from a conditional normalizing flow to teach a feed-forward based flow like SRFlow and WaveGLOW. This work used response-based knowledge distillation, similar to that presented by Hinton et al. This demonstrates that conditional flows can be trained with the standard knowledge distillation techniques, only using the information from the outputs of the teacher and student, but does not demonstrate that such distillation techniques can be used on unconditional flows, invertible flows, nor can feature-based or relation-based knowledge distillation techniques be used.

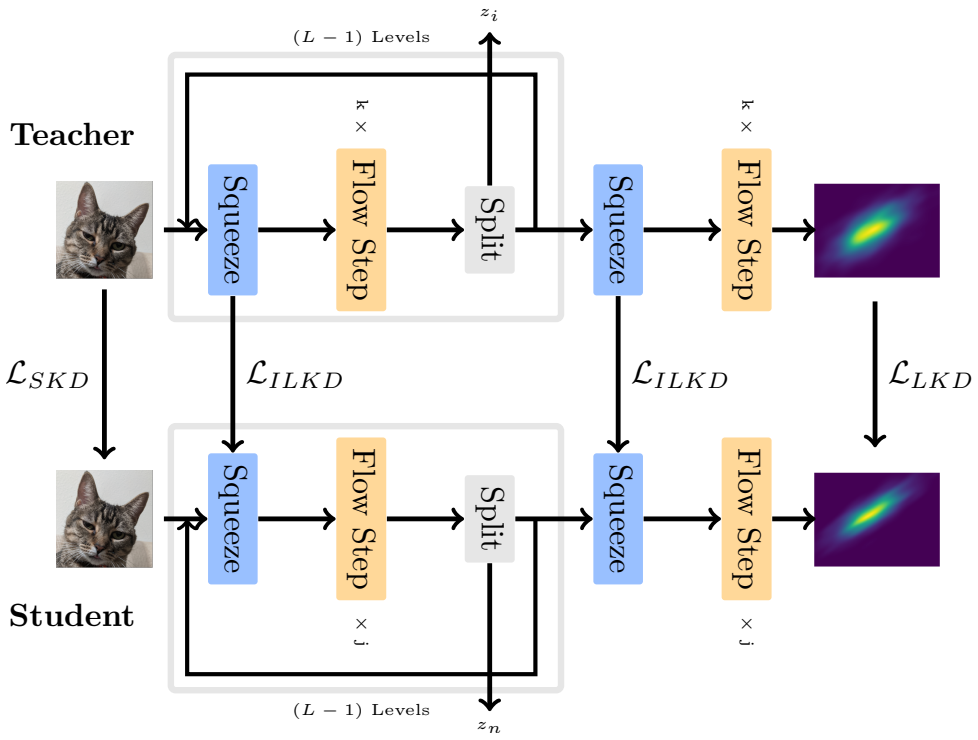


Figure 1. Illustration of different distilling mechanisms using a glow like multi-level architecture. $\mathcal{L}_{SKD}$ shows the synthesized knowledge distillation, $\mathcal{L}_{ILKD}$ shows the intermediate latent knowledge distillation, used at every squeeze step (same as ours) and $\mathcal{L}_{LKD}$ shows the latent knowledge distillation. Here our teacher has k steps (depth) of flow and our student has j, where $j < k$. Note that they both have the same number of levels.

2.3. Probability Density Distillation

Probability Density Distillation, PDD, was first introduced by van den Oord *et al*. [24] as a potential solution to WaveNet's [33] poor sampling performance, thus it became the de facto standard in the waveform modulation due to the significant increase in sampling rates.

The original WaveNet (WN) model is autoregressive, which benefits from the PDD training process, as the model can efficiently run in parallel due to its convolutional nature. However, while generating samples, input features for each layer must be first drawn from the previous feature layer, making parallel inference impossible.

The PDD method aims to translate learned distribution from an easy-to-train autoregressive model to the easy-to-parallelize Inverse Autoregressive Flow (IAF) [14]. Since both of the described models can do density estimation, the proposed way to distillate is to constrain the IAF's output to match the pre-trained WaveFlow model:

$$D_{\mathrm{KL}}(P_{\mathrm{IAF}} \| P_{\mathrm{WN}}) = H(P_{\mathrm{IAF}} \| P_{\mathrm{WN}}) + H(P_{\mathrm{IAF}}) \qquad (11)$$

where $H(P_{\mathrm{IAF}} \| P_{\mathrm{WN}})$ and $H(P_{\mathrm{IAF}})$ denotes cross-entropy and entropy respectively.

The PDD method quickly became a baseline solution for training small flows for waveforms modulation and inspired a variety [27, 28] of other works. Recent work presented by Hoogeboom et al. [12] takes a different approach by incorporating techniques from diffusion models to also increase efficiency.

3. Method

Normalizing flows are unique in their structure due to the composition of composable diffeomorphic functions. This means that we can transform information in many unique ways as compared to other types of deep neural networks. Because normalizing flows operate by a change of variables transformation, we can assume certain properties at different depths of the network. Namely, if we use different levels, in a hierarchical structure, like glow, we may want the student and teacher to share similar latent distributions at those points. Similarly, since we are able to model probability density functions, we know that we want the student and teacher to share this same information. We discuss these techniques in the following subsection. Additionally, we note that these techniques are not limited to feed-forward students nor do they require conditional inference. Thus

these techniques extend the work of Baranchuk *et al*. [2]

3.1. Latent Knowledge Distillation

A basic version of knowledge transfer for normalizing flows is to teach the student's latent distribution to be similar to the teacher's. This is known as *Latent Knowledge Distillation* (LKD). Loss can be calculated between two latent vector representations, as shown in equation 12. This loss may be arbitrarily, but we use a L_1 loss for our experiments.

$$\mathcal{L}_{\text{LKD}}(t, s, x) = \mathcal{L}_r(t(x), s(x)) \tag{12}$$

where t represents the teacher and s represents the student, and x is a sample ($x \sim x$) from the dataset.

While it is a possible approach to do a knowledge distillation on normalizing flows, it has a number of serious drawbacks. First of all, distilling in this manner we would be limited to our dataset, meaning that we can only pass information to the smaller model on already known locations. Another issue with this approach is that it is not making use of flow's invertible property, effectively treating a very complex model as a simple regression. While this is useful, we are not transferring as much knowledge as is possible from the teacher to the student.

3.2. Intermediate Latent Knowledge Distillation

An extension to latent knowledge distillation is to use latent information from intermediate features of the flow. Similar to the perceptual knowledge distillation [35], one could link intermediate layers between the teacher and student flow blocks. While this approach works just like the previous one, it has a nice benefit of constraining not only output but a student flow as a whole, making knowledge distillation "stronger." Through this method we are able to pass significantly more information from the teacher network to the student. We refer to this type of knowledge transfer as *Intermediate Latent Knowledge Distillation* loss, or ILKD.

For an arbitrary flow step we can compare the latent information between the student and teacher, similar to equation 12. The difference is that this time we are using an arbitrary flow step. We will refer to the teacher step as t_i and the student step as s_i. Similarly the metric may be arbitrary, but we use L_1.

$$\mathcal{L}_{\text{ILKD}}(t, s, x) = \sum_i \mathcal{L}_r(t_i(x), s_i(x)) \tag{13}$$

While many networks have explored the usage of L_p losses in their networks, Compositional Normalizing Flows have a unique property that subtly differentiate this formulation. At each layer the network must describe a probability distribution and since each transformation is a change of variables, this results in transformations that do not contain knots in their trajectories [3, 4, 8]. L_p losses calculate the differential between individual elements in the latent representation, they place pressure on the networks to have the same geometric representations within the latent space. Due to the aforementioned properties of compositional Normalizing Flows, when the L_p loss is minimized, so will distributional losses like KL-Divergence. We use L_1 loss due to the high dimensional nature of our data and the purpose of ILKD is to ensure that the trajectories of the teacher and student models align. The goal is to compress N flow steps into M, where $M < N$.

While we could use any arbitrary flow step to distill knowledge to, there are more optimal steps to transfer to. We choose to use flow steps that logically correspond to the progressive change of variables. One simple example of this may be to have a student model have half the depth of the teacher and then every student flow step learns from every other teacher flow step. This example would be akin to trying to having two flow steps in a teacher model distilled into a single flow step in the student model, thus halving the number of flow steps. This ensures alignment within the trajectories and reduces potential complications due to the biases of different flow step formulations. Within our work we choose to distill this knowledge at each split level, finding that this leads to a balance of computation and complexity that each flow can learn. Additionally, this allows for more flexibility, as it is reasonable that the flow steps will need different operations to reach the same latent representation in a different number of steps. Distilling at too frequent of intervals may over constrain the student model, making it inflexible to more complex distributions.

3.3. Synthesized Knowledge Distillation

Another method to transfer knowledge is to look at the synthesized information from each network. Not only do we want the latent information from the flows to be similar, but we also want the generated samples to be similar. To accomplish this simply by using random samples from the learned distribution and then comparing their synthesized results. That is

$$\mathcal{L}_{\text{SKD}}(t, s, z) = \mathcal{L}_r(t^{-1}(z), s^{-1}(z)) \tag{14}$$

here we let z represent the generated sample. Additionally, we use t^{-1} and s^{-1} to represent the inverse flows, which are the generators as described in section 2.1. Unlike a cycle loss [36] we do not need to know the original image that was generated, just that we constrain the student's generation (backwards inference) path to be similar to that of the teacher's. Additionally, unlike Baranchuk *et al*. s distillation technique, we do not require that the generator be conditional.

Since we cannot count on the whole latent space to be as equally covered by the flow mapping, the stability of this method requires that the latent spaces of the student and

Table 1. Averaged test log-likelihood (in nats) for unconditional density estimation (higher is better) across multiple runs.

Architecture	Model	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
GLOW	Student	-0.228	5.967	-22.668	-17.251	147.298
	LKD Student	-0.132	6.008	-22.332	-17.136	162.103
	ILKD Student	-0.133	6.191	-22.187	-17.008	163.148
	SKD Student	$\mathbf{-0.078}$	$\mathbf{6.515}$	$\mathbf{-21.852}$	$\mathbf{-16.130}$	$\mathbf{163.953}$
GLOW	Teacher	0.143	6.604	-19.938	-13.597	165.702
MAF	Student	-0.152	4.385	-21.904	-15.314	155.463
	LKD Student	-0.149	4.473	-21.389	-15.217	155.629
	ILKD Student	$\mathbf{-0.145}$	$\mathbf{4.502}$	$\mathbf{-21.223}$	$\mathbf{-15.184}$	$\mathbf{155.785}$
	SKD Student	-	-	-	-	-
MAF	Teacher	0.133	5.887	-20.662	-13.488	159.442

Table 2. Time consumption for a single batch inference averaged across multiple batches and the number of parameters (in thousands). Average time (ms) and number of parameters (in thousands) are reported.

Arch	Model	Metric	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
GLOW	Student	Time (ms)	2.32 ± 0.16	2.46 ± 0.1	2.55 ± 0.35	2.47 ± 0.07	2.45 ± 0.07
		Params (K)	13.8	14.2	17.4	24.9	34.4
	Teacher	Time (ms)	3.65 ± 0.26	3.88 ± 0.09	4.41 ± 0.28	3.95 ± 0.11	3.89 ± 0.14
		Params (K)	86.7	87.8	96.3	114.2	134.7
MAF	Student	Time (ms)	2.0 ± 0.21	1.98 ± 0.19	1.82 ± 0.05	1.82 ± 0.05	1.91 ± 0.22
		Params (K)	5.0	5.6	9.4	15.9	21.8
	Teacher	Time (ms)	3.34 ± 0.22	3.22 ± 0.18	3.34 ± 0.23	3.36 ± 0.26	3.45 ± 0.2
		Params (K)	10.1	11.2	18.9	31.8	43.6

teacher are similar to each other. In other words, we should distill the teacher's latent space into the student's one before we start the procedure. We note that we found that this method is often unstable. We believe that this is due to NFs difficulty in backwards inference, notably in that they tend to have poorer sampling quality. We believe that this is still a useful notion and will become more stable as the quality of flow's image generation increases.

3.4. Flow Distillation

we can combine all these types of distillations together and create a much stronger from of distillation that each provides independently. The hyperparameters λ_i, representing the weight of the distillation, the resulting loss can be written as follows:

$$\begin{aligned}\mathcal{L}(t, s, x, z) = &\lambda_0 \log p_s(x) \\ &+ \lambda_1 \mathcal{L}_{\text{(I)LKD}}(t, s, x) \\ &+ \lambda_2 \mathcal{L}_{\text{SKD}}(t, s, z)\end{aligned} \tag{15}$$

here we use $l_{(i)LKD}$ to denote both the intermediate and standard latent knowledge distillation, noting that the final latent step is just another "intermediate" step. We show an example of the full distillation in figure 1.

4. Experiments

All calculations were performed on a single GPU Tesla V100.

4.1. Models

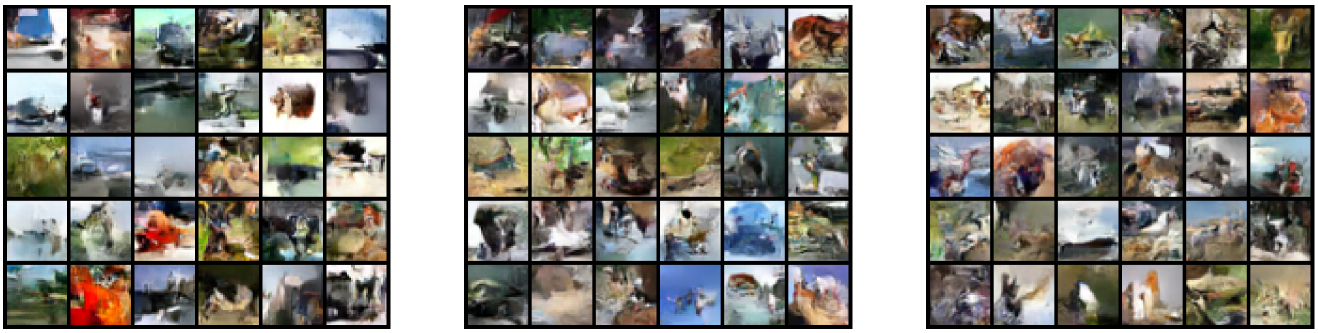
To demonstrate that this method is model agnostic we demonstrate by using Masked Autoregressive Flow [25] and GLOW [13], as previously discussed in Section 2. In all experiments the teacher and students share the same basic model but differ in the number of flow steps, with the student model being smaller than the teacher. For example, in the GLOW model both student and teacher share the number of levels (splits) but have a differing number of flow steps between them. These models have differing architectures but also form the basis of many other types of flow models and thus we believe stand as good proxies.

4.2. Tabular data

We perform density estimation experiments on five standard tabular datasets that include four datasets from the UCI machine learning repository [18] and on a dataset of natural image patches BSDS300 [22]. In Table 1, we report the average log-likelihood on held-out test sets. For each model, we also provide the statistics for time and memory consumption in table 2.

(a) Teacher (b) With KD (c) Without KD

Figure 2. CelebA samples from teacher model (2a), student model (2b), and student model with no knowledge distillation (2c). All images are generated at 64×64 resolution and with temperature=0.7.



(a) Teacher (b) With KD (c) Without KD

Figure 3. CIFAR-10 samples from teacher model (3a), student model (3b), and student model with no knowledge distillation (3c). All images are generated at 32 × 32 resolution and with a temperature of 0.7.

As can be seen in table 2, the proposed LKD and the improved ILKD methods both allows for a significant reduction in time and memory consumption. The SKD method not only improves student flow performance on all datasets but also has better quality than other distillation methods for the GLOW teacher and student. However, its numerical instability leads to a discrepancy in the MAF student's training with SKD. We believe that this is due to MAF's weaker modeling power, noting that this model is substantially smaller and significantly under performs compared to the GLOW based models.

We train each flow for 1×10^4 iterations with a batch size of 65,536 elements and a learning rate of 5×10^{-5} on tabular data and AdamW [21] as an optimizer. In Table 2 the rows labeled with LKD we use $\lambda_0 = 1$ ($\lambda_1 = \lambda_2 = 0$), for ILKD we use $\lambda_0 = 0.9$ and $\lambda_1 = 0.1$ ($\lambda_2 = 0$), and for our SKD we use $\lambda_0 = 0.85$ and $\lambda_1 = \lambda_2 = 0.075$, as described in Equation 15. We found that these combinations of weights are stable and improve the quality of distilled models for all datasets. We also note that as with most NFs, training and stability are significantly affected by batch sizes. Additionally, Flow based networks tend to learn best when gradients are not rapidly changed, often requiring longer warmups and gradient clipping. We note here that this is likely a reason that the optimal distillation weights are small for ILKD and SKD experiments. Small perturbations in flow training can often lead to compounding downstream changes. A detailed overview of network parameters can be found in table 3.

Table 3. Model configurations for generation of tabular data. Provided for GLOW and MAF architectures. Number of levels (L) is equal to 1. Notation is taken from the original paper [13].

GLOW	Level (L)	Hidden
Student	3	32
Teacher	3	64
MAF	Depth (K)	Hidden
Student	3	32
Teacher	6	32

4.3. Image data

To demonstrate our method's performance for convolution-based normalizing flows, we used CelebA [20] and CIFAR-

10 [16] image datasets. In the following experiments, we used GLOW [13] for the unconditional image generation. Both of these experiments use the affine coupling layer as discussed in the GLOW paper. Here we use the same training parameters as the tabular data but with a batch size of 32.

We found that the SKD increased the quality of the student models, but also found that this method was unstable. We, again, believe that this is due to the low sampling quality of these networks. Because of this we focus on just using ILKD, as this is both stable and consistently increases the quality of generated samples.

Configurations of trained models are provided in table 5. Additionally, Figures 2 and 3 show random samples from the teacher models as well as students with and without knowledge distillation. We can see that the students with knowledge distillation significantly outperform students without knowledge distillation. This reflects the results we saw in the table 4. Here we note, as seen in Table 5 that the CIFAR student is a quarter the size of the teacher yet has a $< 2\%$ difference in sampling quality, an improvement of 3% over the non-distilled student's FID. Similarly, the CelebA's student is approximately one eighth of the teacher model and over 35% improvement over the baseline student. This directly demonstrates that these distillation techniques have a powerful effect on student models.

Table 4. Metrics for the image generation task for the GLOW architecture using ILKD on the test set: bits per dimension and FID (lower is better).

	CIFAR-10		CelebA	
	bpd	FID	bpd	FID
Student	3.498	71.177	2.479	68.127
ILKD Student	**3.481**	**69.371**	**2.475**	**54.480**
Teacher	3.423	68.503	2.474	37.460

Table 5. Model configurations for CIFAR-10 image generation tasks (GLOW). Notation is taken from the original paper [13].

	Levels (L)	Depth (K)	Hidden	Params
CIFAR-10				
Student	8	3	512	11M
Teacher	32	3	512	44.2M
CelebA				
Student	16	3	256	7.9M
Teacher	32	3	512	61.2M

4.4. Latent Space Corruption

To ensure the knowledge distillation does not corrupt the hidden space, we need to ensure that random samples from the students still maintain similar quality images. With high dimensional information, it is possible for Normalizing Flows, and other models, to have a small KL-Divergence but also have poor sampling quality. However, high quality samples can only happen if there is a sufficiently good enough cover within the learned latent space. Thus, we propose to measure the quality of the inferred samples for randomly chosen images $\boldsymbol{u}, \boldsymbol{v}$ and an $\alpha \in [0, 1]$. The preserved norm of the latent vector can be defined as:

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{\alpha}) = ((1-\alpha)\boldsymbol{f}(\boldsymbol{u}) + \alpha \boldsymbol{f}(\boldsymbol{v})) \\ \cdot \frac{(1-\alpha)||\boldsymbol{f}(\boldsymbol{u})|| + \alpha||\boldsymbol{f}(\boldsymbol{v})||}{||(1-\alpha)\boldsymbol{f}(\boldsymbol{u}) + \alpha \boldsymbol{f}(\boldsymbol{v})||} \end{aligned} \tag{16}$$

The results of this method are provided for CelebA dataset in Table 6. In this table we can see that the ILKD Student performs significantly better than the student without knowledge distillation. This is especially true for a temperature of 0.7, which generates better samples on all models. A similar temperature was found to have better performance in the original GLOW paper. In Figures 2 and 3 we can also see that the distilled models produce significantly higher quality samples than the non-distilled student models. We show the resultant FID scores for these different temperatures in Table 6. We note here the significant improvements

Table 6. CelebA FID values of images obtained by interpolation in the latent space of trained models.

	FID	
	temp 1.0	temp 0.7
Student	40.159	28.432
ILKD Student	**28.413**	**19.688**
Teacher	19.062	16.382

5. Conclusion

In this work, we demonstrated a novel methods for normalizing flow knowledge distillation, taking advantage of their unique properties. We demonstrated that on a variety of different types of datasets that the methods significantly improve the performance of every flow, greatly decreasing the sampling quality gap between teacher and student flows.

Compared to other distillation methods, ours utilize the unique properties of the normalizing flows' invertibility for better quality and performance. This allows for high quality students to be trained in a simple and efficient manner with minimal loss to sampling performance. Additionally, the resulting probability properties of a distilled flow are kept, making our method a straightforward application to normalizing flow distillation.

References

[1] Sanjeev Arora, Andrej Risteski, and Yi Zhang. Do GANs learn the distribution? some theory and empirics. In *International Conference on Learning Representations*, 2018. 1

[2] Dmitry Baranchuk, Vladimir Aliev, and Artem Babenko. Distilling the knowledge from conditional normalizing flows. In *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*, 2021. 3, 5

[3] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018. 2, 5

[4] Ricky T. Q. Chen, Jens Behrmann, David K Duvenaud, and Joern-Henrik Jacobsen. Residual flows for invertible generative modeling. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019. 5

[5] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp, 2016. 2, 3

[6] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation, 2015. 1

[7] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. 1

[8] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models, 2018. 5

[9] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 2, 3

[10] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, pages 6840–6851. Curran Associates, Inc., 2020. 2

[11] Emiel Hoogeboom, Jorn W. T. Peters, Rianne van den Berg, and Max Welling. Integer discrete flows and lossless compression, 2019. 3

[12] Emiel Hoogeboom, Alexey A. Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models, 2021. 4

[13] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions, 2018. 2, 6, 7, 8

[14] Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improving variational inference with inverse autoregressive flow, 2016. 2, 4

[15] Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979, 2021. 2

[16] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Univeristy of Toronto, 2009. 8

[17] Yann LeCun and Fu Jie Huang. Loss functions for discriminative training of energy-based models. In *International Workshop on Artificial Intelligence and Statistics*, pages 206–213. PMLR, 2005. 1

[18] M. Lichman. Uci machine learning repository, 2013. 6

[19] Alexandra Lindt and Emiel Hoogeboom. Discrete denoising flows, 2021. 3

[20] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015. 7

[21] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. 7

[22] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, pages 416–423, 2001. 6

[23] Vaishnavh Nagarajan. Theoretical insights into memorization in gans. 2019. 1

[24] Aaron Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, et al. Parallel wavenet: Fast high-fidelity speech synthesis. In *International conference on machine learning*, pages 3918–3926. PMLR, 2018. 4

[25] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation, 2017. 2, 6

[26] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference, 2021. 2

[27] Wei Ping, Kainan Peng, and Jitong Chen. Clarinet: Parallel wave generation in end-to-end text-to-speech. *arXiv preprint arXiv:1807.07281*, 2018. 4

[28] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621. IEEE, 2019. 4

[29] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows, 2015. 1, 2

[30] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets, 2015. 3

[31] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015. 1

[32] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021. 2

[33] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016. 4

[34] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A gen-

erative model for raw audio. *CoRR*, abs/1609.03499, 2016. 1

[35] Lucas D. Young, Fitsum A. Reda, Rakesh Ranjan, Jon Morton, Jun Hu, Yazhu Ling, Xiaoyu Xiang, David Liu, and Vikas Chandra. Feature-align network with knowledge distillation for efficient denoising, 2021. 5

[36] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017. 5