# Deep Density Destructors

**David I. Inouye** [1] **Pradeep Ravikumar** [1]

## Abstract

We propose a unified framework for deep density models by formally defining *density destructors*. A density destructor is an invertible function that transforms a given density to the uniform density—essentially destroying any structure in the original density. This destructive transformation generalizes Gaussianization via ICA and more recent autoregressive models such as MAF and Real NVP. Informally, this transformation can be seen as a generalized whitening procedure or a multivariate generalization of the univariate CDF function. Unlike Gaussianization, our destructive transformation has the elegant property that the density function is equal to the absolute value of the Jacobian determinant. Thus, each layer of a deep density can be seen as a shallow density—uncovering a fundamental connection between shallow and deep densities. In addition, our framework provides a common interface for all previous methods enabling them to be systematically combined, evaluated and improved. Leveraging the connection to shallow densities, we also propose a novel tree destructor based on tree densities and an image-specific destructor based on pixel locality. We illustrate our framework on a 2D dataset, MNIST, and CIFAR-10. Code is available on first author's website.

## 1. Introduction

Creating a complex model can be viewed from two perspectives: one constructive and one destructive. As an illustration, suppose someone is trying to build a replica of a Lego model with a set of the same pieces. From a constructive perspective, the person could build an entire replica, compare it to the original model, rebuild the replica based on the comparison, and repeat until the replica is similar to the original model. From a destructive perspective, the person could remove one piece at a time from the original model and record where each piece was removed; then, the person could easily construct a replica by reversing the recorded destructive process.

Analogous to building a replica given an original model, estimating a complex distribution via deep networks given samples from the true distribution can be viewed from the constructive or destructive perspective. Deep generative models such as VAEs (Kingma & Welling, 2014; Rezende et al., 2014), GANs (Goodfellow et al., 2014) and variants estimate a *constructive* transformation from samples of a simple base distribution to samples from a complex distribution (e.g. distribution of real images).[1] Formally, this amounts to the following approximation:

$$\begin{aligned} \boldsymbol{z} &\sim \text{BaseDistribution} \\ \widehat{\boldsymbol{x}} \equiv G_\phi(\boldsymbol{z}) &\sim \text{GenerativeDistribution}\,, \end{aligned} \tag{1}$$

where $\boldsymbol{z}$ is a latent variable, $\phi$ are the parameters of the generative network, and $\widehat{\boldsymbol{x}}$ approximates $\boldsymbol{x}$, the true observed variable. As a mirror of the constructive process, we propose estimating an *invertible* destructive transformation, or destructive *flow*, from the input distribution to the base distribution:

$$\begin{aligned} \boldsymbol{x} &\sim \text{DataDistribution} \\ \widehat{\boldsymbol{z}} \equiv D_\theta(\boldsymbol{x}) &\sim \text{ApproximateBaseDistribution}\,, \end{aligned} \tag{2}$$

where $\boldsymbol{x}$ is from the true data distribution and $\widehat{\boldsymbol{z}}$ is an approximation to the base distribution of $\boldsymbol{z}$. The destructive transformation $D_\theta$ is similar to the inference network in VAEs (Kingma & Welling, 2014; Rezende et al., 2014) but while VAEs only use an (non-invertible) inference network for learning the generative model $G_\phi$, the destructive perspective *implicitly* estimates a generative transformation via the inverse of $D_\theta$, namely $G_\phi \equiv D_\theta^{-1}$ in Eq. 1. Thus, learning simplifies to only estimating a single deep network $D_\theta$ rather than two. Note that if the destructive model is flexible enough so that $\widehat{\boldsymbol{z}} = \boldsymbol{z}$, then $\widehat{\boldsymbol{x}} = \boldsymbol{x}$ because the network is invertible. The beauty of density estimation via invertible transformations is that not only can the model be sampled from, but the likelihood values can be computed

[1]Machine Learning Department, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA. Correspondence to: David I. Inouye <dinouye@cs.cmu.edu>.

*Proceedings of the $35^{th}$ International Conference on Machine Learning*, Stockholm, Sweden, PMLR 80, 2018. 

[1]The transformation can be stochastic as in VAEs or deterministic as in GANs.

exactly unlike in VAEs or GANs; the likelihood of invertible models can be computed based simply on the following change of variables formula:

$$\mathbb{P}_{\widehat{\boldsymbol{x}}}(\boldsymbol{x}) = \mathbb{P}_{\boldsymbol{z}}(D_\theta(\boldsymbol{x}))\,|\det J_{D_\theta}|\ , \tag{3}$$

where $J_{D_\theta}$ is the Jacobian of $D_\theta$.[2]

Estimating deep invertible transformations is not a new idea as evidenced by the Gaussianization line of work (Chen & Gopinath, 2000; Lin et al., 2000; Lyu & Simoncelli, 2009; Laparra et al., 2011; Ballé et al., 2016); the core idea can even be traced back to the seminal exploratory projection pursuit paper (Friedman, 1987)[3]. In these papers, some form of multivariate transformation was used to make the resulting distribution more and more Gaussian. Then, by sampling from a Gaussian and computing the inverse transformation, a sample from the original density could be obtained. Tabak & Vanden-Eijnden (2010); Tabak & Turner (2013) propose several specific types of invertible transformations to slowly move the density towards Gaussianity including linear and radial flows. Tabak & Vanden-Eijnden (2010); Tabak & Turner (2013) directly optimize the parameters of the transformations but do not propose an underlying connection to shallow densities as we propose in this paper. More recently, there has been a line of work that carefully constructs a neural network model to parameterize a valid autoregressive distribution (Germain et al., 2015; Dinh et al., 2015; Graves, 2016; Dinh et al., 2017; Papamakarios et al., 2017). These models compute a linear transformation of the inputs but the shift and scale parameters are functions of deep neural networks; thus, these more recent models can be seen as a hybrid between neural networks and linear transformations. Graves (2016) considers the case where the conditionals are mixtures of Gaussians instead of Gaussian as in VAE and proposes a backpropagation algorithm for this case. Papamakarios et al. (2017) propose compositions of these autoregressive models to form deep autoregressive models. One drawback of these recent models is that they are restricted to a particular type of invertible transformation and engineering or combining new types of transformations is not obvious.

We generalize the ideas behind the previous work on invertible transformations and make a fundamental connection to shallow density models. Our framework can leverage the power of both deep and shallow density estimators to form a deep density model. This makes the framework very modular and any advancement in one component, such as a better shallow density estimator, could improve the model as a whole. We will define the few key properties of invertible *density destructors* and show that the absolute value of the Jacobian determinant is exactly the likelihood. Our destructor definition provides a common interface for all previous methods (with trivial modifications as discussed in future sections) enabling them to be systematically combined, evaluated and improved. In addition, our destructive framework can directly incorporate any invertible preprocessing of the input data into the deep model itself rather than requiring independent preprocessing steps such as a logit transformation (Papamakarios et al., 2017) or feature normalization. Given these new definitions, we introduce a novel invertible transformation for tree densities (Ram & Gray, 2011). In addition, we propose an image-specific destructive transformation based on pixel locality to showcase the flexibility of our framework. Merely using shallow density estimates, we train deep density models in a greedy fashion similar to previous work on Gaussianization (e.g. (Tabak & Turner, 2013; Chen & Gopinath, 2000; Laparra et al., 2011)). We believe this abstract framework lays the groundwork for developing modular invertible transformations that can be composed, evaluated, and improved separately or together.

We summarize our key contributions as follows:

1. *Deep density destructor framework* - We define the required key properties of a transformation class corresponding to a class of densities. With this definition, we give an elegant group-theoretic characterization of deep networks in terms of destructors. We show that many previous density models fall under this framework. We propose a greedy and non-parametric layer-wise training algorithm for deep density destructors by leveraging shallow learning algorithms.
2. *Novel destructors enabled by framework* - Given the insights from our framework, we propose a novel tree density destructor based on shallow density estimation trees (DET) (Ram & Gray, 2011), which are density versions of decision trees. In addition, we build regularized linear and Gaussian mixture destructors that implement our destructor interface similar to the previous Gaussianization work and Gaussian mixture distributions (Graves, 2016).[4] In our 2D experiments, we show that regularization of each shallow destructor is critical for good performance on test data. To show that our framework can incorporate prior knowledge, we give an example of an image-specific density destructor based on pixel locality that leverages the smoothness property of images. We implement a greedy and non-parametric layer-wise training algo-

[2]This exact likelihood computation assumes that determinant of the Jacobian of the transformation can be efficiently computed.

[3]While the goal in (Friedman, 1987) was only to find a few interesting directions, the core algorithm of structure removal by univariate Gaussianization was already suggested by (Friedman, 1987).

[4]Note that (Graves, 2016) focused on computing backpropagation gradients for mixture weights rather than leveraging a shallow Gaussian mixture estimator directly as we consider in this paper.

rithm similar to previous Gaussianization work to construct deep density destructors leveraging only shallow learning algorithms from the Python `sci-kit learn` library (Pedregosa et al., 2011) and `mlpack` (Curtin et al., 2013).

**Notation** Let $d$ be the number of features, $n$ be the number of samples, and $k$ be the number of deep layers. Let $F$ and $F^{-1}$ denote the CDF and inverse CDF of a univariate distribution, where $\Phi$ and $\Phi^{-1}$ are the CDF and inverse CDF of the standard univariate normal. Let boldface CDF functions, e.g. $\boldsymbol{F}$, denote independent application of $d$ CDFs: $\boldsymbol{F}_\theta(\boldsymbol{x}) = [F_{\theta_1}(x_1), F_{\theta_2}(x_2), \cdots, F_{\theta_d}(x_d)]$, note that each dimension can have a different $F_{\theta_s}$; let $\boldsymbol{\Phi}(\boldsymbol{x}) = [\Phi(x_1), \Phi(x_2), \cdots, \Phi(x_d)]$ and similarly for $\boldsymbol{\Phi}^{-1}(\boldsymbol{x})$. We denote subvectors using subscripts with MATLAB-like index ranges; for example, $\boldsymbol{x}_{1:3} \in \mathbb{R}^3$ designates the subvector of the first 3 dimensions of $\boldsymbol{x}$.

## 2. Deep Density Destructors

To develop our unified framework, we seek a method for *composing* transformations[5] corresponding to estimated densities, which could include both classical shallow densities and more recent deep densities. In order to accomplish this composition, we propose the following definition for a density destructor class based on a class of densities.

**Definition 1** (Density Destructor Class)**.** *A transformation class* $\{D_\theta \colon \theta \in \Theta\}$ *is said to* destroy *a class of densities* $\{\mathbb{P}_\psi \colon \psi \in \Psi\}$ *if the following properties hold:*

1. *Uniformability*

$$\begin{aligned} &\forall \psi, \ \text{if } \boldsymbol{x} \sim \mathbb{P}_\psi, \\ &\text{then } \exists \theta \text{ s.t. } D_\theta(\boldsymbol{x}) \sim \text{Uniform}([0,1]^d) \end{aligned} \tag{4}$$

2. *Invertibility*

$$\forall D_\theta, \exists D_\theta^{-1} \ \textit{s.t.}\ D_\theta^{-1}(D_\theta(\boldsymbol{x})) = \boldsymbol{x} \tag{5}$$

**Remark 1 on Uniformability Eq. 4** One elegant key property of our destructor definition is that the absolute value of the Jacobian determinant is exactly the likelihood of the corresponding density. This can be easily seen if we let $\boldsymbol{z} \sim$ Uniform in Eq. 3:

$$\mathbb{P}_{\hat{\boldsymbol{x}}}(\boldsymbol{x}) = \mathbb{P}_{\boldsymbol{z}}(D_\theta(\boldsymbol{x}))\,|\det J_{D_\theta}| = |\det J_{D_\theta}|\ . \tag{6}$$

Thus, if a fast method for computing the likelihood exists, the Jacobian matrix can be easily computed—this generalizes the idea from Real NVP (Dinh et al., 2017) and MAF (Papamakarios et al., 2017) where the Jacobian is assumed to be triangular.

**Remark 2 on Uniformability Eq. 4** This novel uniformability property characterizes destructors in an intuitive but general way. Importantly, this novel property essentially transforms a structured density $\mathbb{P}_\psi$ to the least structured density, namely an independent uniform distribution over the hypercube. This builds on the intuition of the univariate CDF, which is a destructive transformation class for any univariate distribution that transforms any variable into a univariate uniform variable. Thus, this requirement can be seen as a generalization of the univariate CDF transformation to a multivariate CDF transformation. Importantly, however, it should be noted that this multivariate CDF *transformation* (where $D_\theta \colon \mathbb{R}^d \to [0,1]^d$) is quite different than the multivariate CDF function (where $\boldsymbol{F} \colon \mathbb{R}^d \to [0,1]$) because it can be inverted unlike the multivariate CDF function. While traditional whitening removes any structure related to the first and second moments via PCA, a destructive transformation can be seen a generalized whitening transformation that removes all structure from the density.

**Remark on Invertibility Eq. 5** Invertibility is required from a destructive perspective so that we can evaluate the generative trasnformation $G_\phi(\boldsymbol{z}) \equiv D_\theta^{-1}(\boldsymbol{z})$ and sample from the estimated distribution via the constructive perspective as in Eq. 1. This invertibility requirement is also the fundamental property of network *flows* (Tabak & Vanden-Eijnden, 2010; Tabak & Turner, 2013; Papamakarios et al., 2017).

**Definition 2** (Canonical Density Destructor Class)**.** *A* canonical *density destructor class is said to destroy a density class if it has the properties in Def. 1 but also has the following properties:*

1. *Canonical domain*

$$\forall \theta, \ D_\theta \colon [0,1]^d \to [0,1]^d \tag{7}$$

2. *Identity element*

$$\exists \theta \ \textit{s.t.}\ D_\theta(\boldsymbol{x}) = \boldsymbol{x} \tag{8}$$

By adding the domain constraint in Eq. 7, we can trivially compose these destructive transformations because the output range of one layer is exactly the input domain of the next layer. The identity requirement in Eq. 8 seems like a modest requirement which merely means that the destructor can model the uniform distribution and is critical to our group-theoretic characterization of deep destructor models given later. In addition, we note that the inverse of a canonical destructor is also a canonical destructor for some (possibly implicit) density—showing the elegant symmetry of the canonical destructor.

[5] Note that combining functions via composition, e.g. $f_k(f_{k-1}(\cdots f_1(\boldsymbol{x})))$ is fundamentally different than combining functions via summation or product, e.g. $\sum_{j=1}^k f_j(\boldsymbol{x})$ or $\prod_{j=1}^k f_j(\boldsymbol{x})$.

**Remark on Unrestricted Nature of Canonical Class** Note that most destructive transformation classes can be nearly trivially mapped to a canonical transformation class by applying an inverse univariate CDF on each dimension, where the CDF's domain matches the original variable's support. For example, given a destructive transformation $D_\theta \colon \mathbb{R}^d \to [0,1]^d$, we can construct $\tilde{D}_\theta(\boldsymbol{u}) = D_\theta(\boldsymbol{\Phi}^{-1}(\boldsymbol{u})) \colon [0,1]^d \to [0,1]^d$. If the original input domain is $\mathbb{R}^d$, we could simply perform the trivial preprocessing step of $\boldsymbol{u} = \boldsymbol{\Phi}(\boldsymbol{x})$. By combining this preprocessing step and modified estimator $\tilde{D}_\theta$, we arrive at the original destructor: $\tilde{D}_\theta(\boldsymbol{u}) = D_\theta(\boldsymbol{\Phi}^{-1}(\boldsymbol{u})) = D_\theta(\boldsymbol{\Phi}^{-1}(\boldsymbol{\Phi}(\boldsymbol{x}))) = D_\theta(\boldsymbol{x})$. Thus, the canonical property does not impose any significant restrictions on the transformation class but makes the transformation class amenable to transformation composition. By restricting the domain to be the unit hypercube, we make all destructors directly interchangeable with each other—thus providing a common interface for all destructors. This canonical density destructor class leads us to the following group-theoretic characterization of destructive deep learning models.

**Definition 3** (Deep Density Destructor Group)**.** *Given a canonical destructive transformation class* $\{D_\theta \colon \theta \in \Theta\}$ *defined in Def. 2, we define the* deep density destructor group *as the group generated by* $\langle \{D_\theta \colon \theta \in \Theta\} \cup \{D_\theta^{-1} \colon \theta \in \Theta\} \rangle$ *where the group operation is function composition.*

The above definition provides an elegant characterization of all possible deep density models of any depth given a destructive transformation class—which can either be defined by the transformation class itself or derived based on a given density class.

## 3. Density Destructor Examples

We show that many previous density models fall under our more general framework and give some new examples of density destructors that are enabled by this new perspective. A summary of example density destructors and their corresponding density models can be seen in Table 1.

We first describe the more recent autoregressive models such as MADE (Germain et al., 2015), NICE (Dinh et al., 2015), Real NVP (Dinh et al., 2017) and MAF (Papamakarios et al., 2017) in terms of our density destructor framework. Under our framework, these models assume that the density is given by the chain rule of probability, namely: $\prod_{s=1}^{d} \mathbb{P}_s(x_s \mid \boldsymbol{x}_{1:s-1})$, where $\boldsymbol{x}_{1:s-1}$ is subvector of all variables up to $s-1$. They parameterize these univariate conditional distributions as a Gaussian or a mixture of Gaussians where the mean, variance and weight parameters are the outputs of neural networks.

Another major class of density destructors, such as Gaussianization (Chen & Gopinath, 2000; Lin et al., 2000; Lyu & Simoncelli, 2009; Laparra et al., 2011; Ballé et al., 2016), is based on linear projections followed by an independent density estimate such as an independent mixture of Gaussians (Chen & Gopinath, 2000) or univariate histograms (Laparra et al., 2011). One key component of these approaches is to find the linear projection that is most independent, namely using a method related to independent components analysis (ICA). Laparra et al. (2011) suggest that possibly even random rotations can be enough in the low dimensional setting though this is unlikely to work in higher dimensions because of the curse of dimensionality.

Another classical model that falls under this framework is copula models (Jaworski et al., 2010). In particular, the Gaussian copula model can be seen as applying a series of independent transforms and a linear rotation based on correlation matrix estimation: $\boldsymbol{\Phi}(R^{-\frac{1}{2}}\boldsymbol{\Phi}^{-1}(\boldsymbol{F}(\boldsymbol{x})))$, where $R$ is the correlation matrix in the transformed space. Note that this could be computed using high-dimensional statistical methods such as the Non-paranormal SKEPTIC (Liu et al., 2012), which may reduce the required sampling complexity. In many ways, our destructor framework is almost like a deep copula model because each density estimate lies on the uniform cube; however, we do not require that the marginal densities be uniform as is the case for copulas.

### 3.1. Gaussian Mixture Model Destructor

Because Gaussian mixture models are often used for shallow density estimators, we developed a density destructor from the Gaussian mixture model density.[6] The key idea is based on the following facts: 1) the conditional distributions of a mixture of Gaussians is also a mixture of Gaussians with modified weights, and 2) the marginal distributions of a mixture of Gaussians is just a univariate mixture of Gaussians. From this, we can directly compute the conditional CDF functions $F_s(x_s \mid \boldsymbol{x}_{\tilde{s}})$, where $\boldsymbol{x}_{\tilde{s}}$ is some set of variables not including $x_s$. This is a special case of an autoregressive model where the conditionals are actually known in closed form. By building this destructor, the power of traditional Gaussian mixture model estimators can be leveraged in our deep density framework.

### 3.2. Density Tree Destructor

A density estimation tree (DET) (Ram & Gray, 2011) is a piecewise constant density defined by a decision tree when the domain is the unit hypercube (or more generally a bounded region). In the original formulation from (Ram & Gray, 2011), each leaf node is associated with a con-

[6] As noted in the introduction, Graves (2016) briefly explored this possibility but focus on backpropagation for learning the mixture weights rather than using shallow density estimates as we do here.

stant density value based on the number of samples that fall into that leaf and the volume of the leaf node. We propose a novel destructor for the tree density that is merely a shift and scaling of the coordinates depending on which leaf node $\mathcal{L}_\ell$ a sample belongs to: $\{\text{diag}(\boldsymbol{a}_\ell)\boldsymbol{x} + \boldsymbol{b}_\ell, \text{ if } \boldsymbol{x} \in \mathcal{L}_\ell\}$, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are based on the density estimate and leaf volume. The set of *disjoint* hyperrectangles defining each leaf node of the input space is mapped to different set of *disjoint* hyperrectangles in the output space such that the new density is uniform. An illustration of a tree destructor can be seen in Fig. 1 and more details can be found in the appendix. We implement both a random tree estimator and

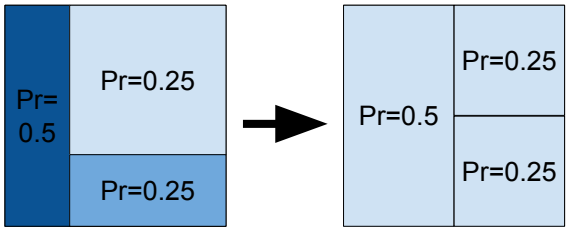


*Figure 1.* Given the initial tree density on the left where Pr indicates the probability of being in that leaf node, the tree destructor resizes the rectangles so that the density is distributed uniformly on the unit hypercube as on the right.

and a wrapper that interfaces with the fast density tree algorithm from (Ram & Gray, 2011) implemented in mlpack (Curtin et al., 2013). Given the tree structure, we then estimate the density of each leaf based on the number of points that fall into the leaf. One straightforward extension is to apply a destructor to each leaf node since each leaf node is independent. For example, an independent histogram could be used to individually destroy each leaf node before projecting to the appropriate hyperrectangle. This emphasizes the modular nature of this framework where destructors can be composed hierarchically.

### 3.3. Image-Specific Feature Pairs

To show that our framework can incorporate prior knowledge, we give an example of a image-specific density destructor that leverages the smoothness property of images. The main idea is that we can destroy the structure based on pixel locality (i.e. the smoothness property of real images) by destroying the structure of pairs of nearby pixels. For this to work, we require that the pairs of pixels be disjoint so that the density model is: $\prod_{P \in \mathcal{P}} \mathbb{P}_P(x_{P(1)}, x_{P(2)})$, where $\mathcal{P}$ is a disjoint set of pixel pairs. For each pair density, we can use any other destructor (including another deep destructor) to destroy the structure in each 2D space; again, this emphasizes the modular and possibly hierarchical nature of our framework since we are free to use any destructor for these pairs. We choose pairs of adjacent pixels rotating between possible directions in one our experiments on the MNIST and CIFAR-10 datasets.

### 3.4. Non-Parametric Greedy Algorithm

We implement a simple non-parametric greedy algorithm which merely estimates a density at each layer, transforms the training data via the associated destructor and repeats this process until the likelihood on a held-out validation set decreases.[7] Note that layers can be added as long as the model does not overfit the training data. For the models in our experiments, we only leverage classical shallow density estimators such as Gaussian mixtures or tree densities at each layer but more advanced density estimators could also be used. In addition, the cumulative log likelihood of the training and validation set is merely the sum of layer log likelihoods because of the uniformability property. Thus, each layer can compute the best density approximation for that layer and can be computed independently of previous layers. Clearly, a non-greedy algorithm that tunes all the previous layers would likely be better but we think this simple greedy algorithm provides a useful baseline. We have found that using highly regularized density estimates at each layer usually performs better because it does not overfit the training data as will be seen in the experiments. As the density estimates approach the uniform distribution, the training data moves less and less at each layer. Thus, similar to simulated annealing, the cooling schedule (i.e. how much regularization is applied to each layer) will affect the quality and depth of the final deep model.

An illustrative example of the progression of our algorithm using a concentric circles dataset and a density tree destructor (described more fully in the experiments section) can be seen in Fig. 2. As the training data is transformed from the original density to the uniform density, the implicit generative density converges towards the true density.

## 4. Experiments

### 4.1. Synthetic Toy Data

We start with some simple 2D experiments to illustrate our density destructor framework and build some intuitions about how to design destructors (code to reproduce this experiment and corresponding figures is available on the first author's website). For this toy experiment, we choose a simple but difficult-to-estimate synthetic dataset of concentric circles as seen in Fig. 2. The radius is a mixture of Gaussians and the angle is uniform. Note that this dataset does not contain any obvious information in the marginals and the underlying density is nearly a low-dimensional manifold—thus very roughly approximating some of the difficult density estimation problems in practice such as

[7] Note that the general greedy approach has been proposed in previous work (e.g. (Tabak & Turner, 2013; Chen & Gopinath, 2000; Laparra et al., 2011)) but previous work did not directly estimate shallow densities at each layer.

*Table 1.* Examples of Density Destructor Classes

| Description | Density | Transformation |
|---|---|---|
| **Autoregressive Density** | $\prod_{s=1}^{d} \mathbb{P}_s(x_s \mid \boldsymbol{x}_{1:s-1})$ | $[F_1(x_1), F_2(x_2 \mid x_1), \cdots, F_d(x_d \mid \boldsymbol{x}_{1:s-1})]$ |
| Mixture of Gaussians Conditionals (e.g. MADE, MAF) | $\prod_{s=1}^{d} \left[ \sum_{t=1}^{m} \pi_t(\boldsymbol{x}_{1:s-1}) \times \mathbb{P}_{\mathcal{N}}(x_s \mid \mu_{st}(\boldsymbol{x}_{1:s-1}), \sigma_{st}^2(\boldsymbol{x}_{1:s-1})) \right]$ | $\left[ F_1(x_1), F_2(x_2 \mid x_1), \cdots, F_d(x_d \mid x_1, \cdots, x_{s-1}) \right]$ |
| Block Gaussian Conditionals (e.g. Real NVP, NICE) | $\mathbb{P}_{\mathcal{N}}(\boldsymbol{x}_{1:t} \mid 0, \mathbf{I}) \times \mathbb{P}_{\mathcal{N}}(\boldsymbol{x}_{t+1:d} \mid \boldsymbol{\mu}(\boldsymbol{x}_{1:t}), \boldsymbol{\sigma}^2(\boldsymbol{x}_{1:t}))$ | $\left[ \Phi(\boldsymbol{x}_{1:t}), \Phi(\frac{x_{t+1} - \mu_{t+1}(\boldsymbol{x}_{1:t})}{\sigma_{t+1}(\boldsymbol{x}_{1:t})}), \cdots, \Phi(\frac{x_d - \mu_d(\boldsymbol{x}_{1:t})}{\sigma_d(\boldsymbol{x}_{1:t})}) \right]$ |
| **Linear Projection Density** | $\mathbb{P}_{\psi}(W\boldsymbol{x})$ | $D_{\theta}(W\boldsymbol{x})$ |
| Independent Components (e.g. Gaussianization via ICA) | $\prod_{s=1}^{d} \mathbb{P}_s(\boldsymbol{w}_s^T \boldsymbol{x})$ | $\boldsymbol{F}(W\boldsymbol{x})$ |
| Gaussian (e.g. via PCA) | $\mathbb{P}_{\mathcal{N}}(\boldsymbol{x} \mid \boldsymbol{\mu}, \Sigma)$ | $\boldsymbol{\Phi}(\Sigma^{-\frac{1}{2}}(\boldsymbol{x} - \boldsymbol{\mu}))$ |
| **Copula-based Density** | $\mathbb{P}^{\text{cop}}(\boldsymbol{F}(\boldsymbol{x})) \prod_{s=1}^{d} \mathbb{P}_s(x_s)$ | $D_{\theta}(\boldsymbol{F}(\boldsymbol{x}))$ |
| Gaussian Copula | $\mathbb{P}_R^{\mathcal{N}\text{-cop}}(\boldsymbol{F}(\boldsymbol{x})) \prod_{s=1}^{d} \mathbb{P}_s(x_s)$ | $\boldsymbol{\Phi}(R^{-\frac{1}{2}} \boldsymbol{\Phi}^{-1}(\boldsymbol{F}(\boldsymbol{x})))$ |
| **Gaussian Mixture** (note that $F_s(x_s \mid \boldsymbol{x}_{-s})$ is computable) | $\sum_{t=1}^{m} \pi_t \mathbb{P}_{\mathcal{N}}(\boldsymbol{x})$ | $[F_1(x_1), F_2(x_2 \mid x_1), \cdots, F_d(x_d \mid x_1, \cdots, x_{s-1})]$ |
| Examples of new destructors enabled by our unified destructor framework | | |
| **Piecewise Density (or Tree Density)** | $\{\mathbb{P}_{\psi_\ell}(\boldsymbol{x}), \text{ if } \boldsymbol{x} \in \mathcal{L}_\ell\}$, where $\mathcal{L}_\ell$ are the disjoint subspaces of the leaves. | $\{D_{\theta_\ell}(\boldsymbol{x}), \text{ if } \boldsymbol{x} \in \mathcal{L}_\ell\}$ |
| Piecewise Uniform (e.g. DET) | $\{c_\ell, \text{ if } \boldsymbol{x} \in \mathcal{L}_\ell\}$ | $\{\text{diag}(\boldsymbol{a}_\ell)\boldsymbol{x} + \boldsymbol{b}_\ell, \text{ if } \boldsymbol{x} \in \mathcal{L}_\ell\}$ |
| **Image-Specific Feature Pairs** | $\prod_{P \in \mathcal{P}} \mathbb{P}_P(x_{P(1)}, x_{P(2)})$, where feature pairs $\mathcal{P}$ are based on pixel locality. | $\{D_P(x_{P(1)}, x_{P(2)}), \forall P \in \mathcal{P}\}$ |

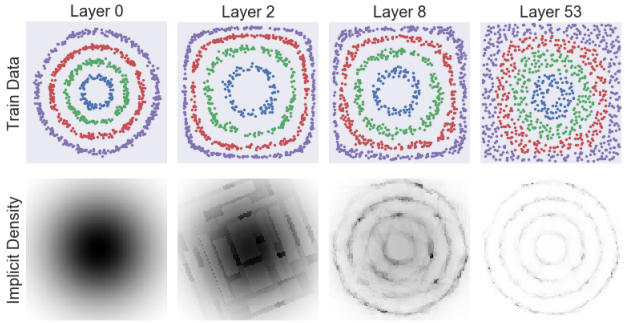


*Figure 2.* The transformed samples (top) and implicit density (bottom) at different layers of the DensityTree (100) model described in Sec. 4. While the initial layers do not provide a good estimate of the underlying density, after many layers, the underlying density begins to model the true underlying patterns. (Enlarged figure in appendix.)

image densities. We now describe several baseline shallow models, and representative deep models from the linear destructor class, the Gaussian mixture destructor class and the tree destructor class as outlined in the next paragraphs. Note that we merely use an independent Gaussian destructor as the first destructor for all the non-baseline models to project the data into the canonical unit hypercube space.

**Baseline Models** We choose a multivariate Gaussian, a mixture of twenty spherical Gaussians and two tree densities: (1) a random tree density with the maximum number of leaves set to 50 and the density of each leaf node is approximated by independent histograms with 10 bins, and (2) a density tree where the min samples per leaf is 10. For the histograms, we regularize the bin counts by adding $\alpha = 10$ pseudo-counts to each bin, which can be seen as putting a Dirichlet prior on the histogram bins.

**Linear Destructor** For simplicity, we choose a random projection model where the data is orthogonally projected onto a new space similar to (Laparra et al., 2011). We build a canonical destructor by composing the independent standard normal inverse CDF, a random linear projection, a standard normal CDF (which returns the values to the unit hypercube) and finally an independent histogram on the unit hypercube: $\boldsymbol{F}_{\text{hist}}(\boldsymbol{\Phi}(A_{\text{rand}} \boldsymbol{\Phi}^{-1}(\boldsymbol{x})))$. The histogram was estimated with 20 bins and a regularization parameter (pseudo-counts) $\alpha$ in the set $\{0.1, 1, 10\}$. Unlike in (Laparra et al., 2011) where the histograms are estimated on an unbounded domain, we project our data onto the bounded unit hypercube via the normal CDF before estimating the histograms so that we do not have to handle histogram boundary issues—because histograms are inherently bounded density estimates. We allow $\alpha$ to vary to

show the effect of regularization on the depth and performance of the deep destructor. Note that as $\alpha \to \infty$ the estimate approaches the uniform distribution.

**Gaussian Mixture Destructor** We merely compose two transformations to form a canonical Gaussian mixture destructor: 1) an inverse normal CDF transformation—this projects from the unit hypercube onto the unbounded real space—, and 2) a spherical Gaussian mixture density destructor, which can now be estimated in an unbounded space because of the first transformation. In our preliminary experiments, we found that an unregularized Gaussian mixture density destructor often performed poorly. Thus, we developed a simple variation in which a mixture model is first estimated. Then, a standard multivariate normal component (i.e. $\mathcal{N}(0, \boldsymbol{I})$) is added with a fixed mixture weight $w \in \{0.1, 0.5, 0.9\}$. Finally, the density is refit with the new component fixed. Thus, this smooths the mixture density towards a standard multivariate normal, and thus, $w$ is a simple regularization parameter.

**Random Tree Destructor** For our random tree destructor, we first apply a random rotation to avoid the effects of a single coordinate view. Then, we construct a tree density using random splits with a maximum number of nodes set to 4 and a minimum leaf size set to 10% of the training data—note that this is a highly regularized tree since there can only be 4 leaf nodes. At each leaf node, we apply a histogram destructor with 10 bins and let the regularization parameter $\alpha$ vary in the set $\{1, 10, 100\}$. This destructor illustrates the power of creating complex destructors from simpler destructors.

**Density Tree Destructor** Similar to our random tree destructor, we first apply a random rotation. Then, we estimate a full tree density using the DET (Ram & Gray, 2011) estimator implemented in `mlpack` (Curtin et al., 2013) where we set the minimum number of samples per leaf to 10. To regularize the density tree, we implicitly estimate a smoothed tree by taking a mixture of the empirical density tree estimate, which is based on leaf counts, and the uniform distribution with weight $w \in \{0.1, 0.5, 0.9\}$ similar to the mixture weight used in the Gaussian mixture destructor. If $w = 1$, then the estimated density is exactly the uniform whereas if $w = 0$, the estimated density is solely based on normalizing the empirical leaf counts.

The geometric mean of the test likelihood (i.e. the exponential of the mean log likelihood) along with the selected number of layers for the concentric circles dataset can be seen in Fig. 3. Clearly the deep models outperform their shallow counterparts. Note that the deep models also provide a useful latent space that is not possible with shallow models. We observe that as expected, the number of layers increases as the regularization increases. The best model is a random tree with a very high regularization of $\alpha = 100$ and a very deep network of 172 layers. We note, however, that the best density tree model only has 53 layers and is thus a strong but much more compact model. These results give evidence for the idea that composing many highly regularized transformations is better than composing a small number of unregularized transformations. However, we also note that more layers does not always translate into better performance as the case for the Gaussian mixture models where $w = 0.5$ with 16 layers outperforms $w = 0.9$ with 36 layers.

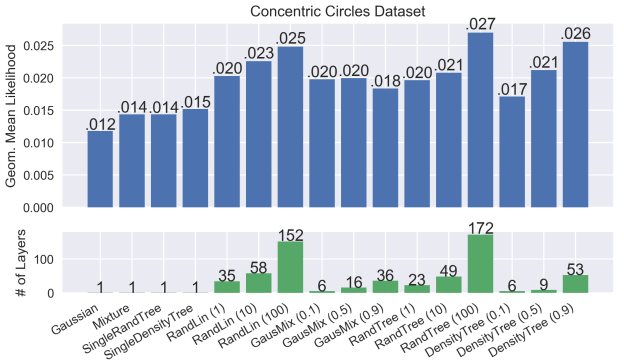


*Figure 3.* Deep models clearly outperform baseline models, and random trees seem to perform the best in terms of test likelihood. As expected, the selected number of layers increases as the amount of regularization on each layer increases. (Enlarged figure in appendix.)

We show train and test final transformations (i.e. in the latent space) for the best performing models in each class in Fig. 4. As a first observation, we notice that the test transformation is clearly less uniform than the train transformation due to finite sample errors. While the Gaussian mixture with $w = 0.5$ performs reasonably in terms of log likelihood, the transformation is clearly irregular with many bumps in comparison to the original data; this is likely due to the fact that a Gaussian mixture model is parametric rather than non-parametric and can only model elliptical regions. In light of this discrepancy, it would be interesting to explore other quality measures other than test likelihood but that is out of the scope of this paper. For the linear model, it is interesting to note that the edges (i.e. the outermost circle) is the most uniformly distributed; this effect is likely due to the fact that the inner structure is somewhat obscured in marginal distributions in comparison to the outer structure. For both tree models, the latent space transformation maintains the circular structure of the original data which suggests that highly regularized trees may be an good general destructor at least in small dimensions. Comparing both tree models, we notice that while they are very similar in spreading the training points out evenly, the density tree is slightly better at maintaining the underlying

circle structure. For example notice the slightly distorted right section of RandTree (100).

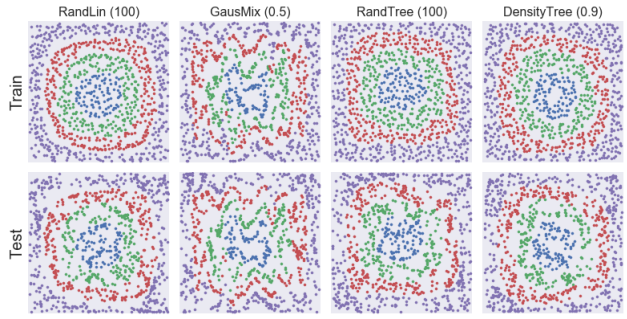


*Figure 4.* Each deep model transforms the data differently with DensityTree seeming to be the most regular. Notice that while the train data may be evenly spaced on the unit square, this does not mean the test data is also uniform. This is why highly regularized destructors are required to build usable deep models via a greedy algorithm. (Enlarged version of figure in the appendix.)

## 5. Image Experiments

We now give some initial results using the MNIST and CIFAR-10 datasets to show that it is possible to train density destructor models on larger datasets. We base our experiments on the unconditional (i.e. unsupervised) MNIST and CIFAR-10 experiments in (Papamakarios et al., 2017) and use the same preprocessed data as in (Papamakarios et al., 2017). As with the 2D experiment, code is available to reproduce the estimation of our deep density models.

The preprocessed data in (Papamakarios et al., 2017) projected the data into the unbounded data space via the logit transform. Because we can model the data in a bounded region, we first apply an inverse logit preprocessing transformation to return the data to the original space (the change in likelihood is directly accounted for because preprocessing steps can be directly incorporated in our destructive framework). We first choose a deep Gaussian copula model which just applies a Gaussian copula destructor at each layer; we use a histogram destructor with 40 bins and $\alpha = 100$ for the marginals. We also apply our image pairs destructor by pairing adjacent pixel coordinates—rotating through the 8 possible directions. For the pair destructor, we evaluate both a Gaussian copula destructor and a single density tree with the minimum number of samples per leaf set to 100 and the maximum number of leaf nodes set to 50. We compare to the state-of-the-art models in (Papamakarios et al., 2017) and show the test log likelihood, depth, and training time in Table 2. Note that the timings for the baselines from (Papamakarios et al., 2017) are based on using a Titan X GPU whereas our methods merely use at most 10 CPUs. We have not attempted to optimize our code for GPU usage but think this is an excellent future direction.

For MNIST, the deep copula model and the image pairs (tree) models outperform previous models including MAF. The experiments on CIFAR-10 illustrate both the power and the limitations of the particular destructors we selected. The deep copula model can model some of the massive dependencies in real images which allows it to surpass all but the MAF models in performance. Note that the first layers of the deep copula have the following log likelihood values (-4572, 1881, 2368, 2461)—notice the massive increase of log likelihood within the first several layers. On the other hand, the image pairs estimators are not powerful enough and stop in a very bad minimum because they can only model a small amount of dependency in a small local region. While CIFAR-10 shows the limitation of the particular instantiations we selected for this experiment, we have proposed a general framework for building new destructors that would enable composing a deep copula destructor and then applying a tree destructor.

*Table 2.* Test Log Likelihood, Depth and Train Time

| | MNIST | | | CIFAR-10 | | |
|---|---|---|---|---|---|---|
| | LL | D | T | LL | D | T |
| *Models from MAF paper computed on Titan X GPU* | | | | | | |
| Gaussian | -1367 | 1 | 0.0 | 2367 | 1 | 0.0 |
| MADE | -1385 | 1 | 0.0 | 448 | 1 | 0.2 |
| MADE MoG | -1042 | 1 | 0.1 | -53 | 1 | 0.3 |
| Real NVP | -1329 | 5 | 0.2 | 2600 | 5 | 1.4 |
| Real NVP | -1765 | 10 | 0.2 | 2469 | 10 | 1.0 |
| MAF | -1300 | 5 | 0.1 | 2941 | 5 | 3.7 |
| MAF | -1314 | 10 | 0.2 | **3054** | 10 | 7.5 |
| MAF MoG | -1100 | 5 | 0.2 | 2822 | 5 | 3.9 |
| *Our proposed destructors computed on 10 CPUs* | | | | | | |
| Copula | -1028 | 5 | 0.2 | 2626 | 17 | 10.1 |
| Pairs (Cop) | -1043 | 17 | 0.7 | -2518 | 31 | 7.4 |
| Pairs (Tree) | **-1003** | 21 | 1.0 | -2404 | 31 | 38.0 |

## 6. Conclusion

We introduced a density destructor framework that enables modular creation of deep density models by composing density models (whether shallow or deep) and their corresponding density destructors. We believe this work opens up many more possibilities for the principled design and analysis of unsupervised deep networks that leverages the insight, algorithms and intuitions behind classical shallow density models rather than having to work with black box models. In addition, our framework may help understanding traditional deep models by reinterpreting them from a destructor perspective and finding the implicit densities of each layer. Thus, we hope to lay the groundwork for many interesting research directions based on the density destructor framework.

## Acknowledgements

We acknowledge the support of NSF via IIS-1149803, IIS-1664720, DMS-1264033, and NIH via R01 GM117594-01 as part of the Joint DMS/NIGMS Initiative to Support Research at the Interface of the Biological and Mathematical Sciences.

## References

Ballé, J., Laparra, V., and Simoncelli, E. P. Density modeling of images using a generalized normalization transformation. *ICLR*, pp. 1–12, 2016.

Chen, S. S. S. and Gopinath, R. A. Gaussianization. *NIPS*, pp. 423–429, 2000.

Curtin, R. R., Cline, J. R., Slagle, N. P., March, W. B., Ram, P., Mehta, N. A., and Gray, A. G. MLPACK: A scalable C++ machine learning library. *Journal of Machine Learning Research*, 14:801–805, 2013.

Dinh, L., Krueger, D., and Bengio, Y. NICE: Non-linear independent components estimation. In *arXiv preprint arXiv:1410.8516*, 2015.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real NVP. In *ICLR*, 2017.

Friedman, J. H. Exploratory projection pursuit. *Journal of the American Statistical Association*, 82397:249–266, 1987.

Germain, M., Gregor, K., Murray, I., and Larochelle, H. MADE: Masked autoencoder for distribution estimation. In *ICML*, 2015.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *NIPS*, pp. 2672–2680, 2014.

Graves, A. Stochastic backpropagation through mixture density distributions. *arXiv:1607.05690v1*, 2016.

Jaworski, P., Durante, F., Härdle, W., and Rychlik, T. (eds.). *Copula Theory and Its Applications*. Springer, 2010.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *ICLR*, 2014.

Laparra, V., Camps-Valls, G., and Malo, J. Iterative Gaussianization: From ICA to random rotations. *IEEE Transactions on Neural Networks*, 22(4):537–549, 2011.

Lin, J., Saito, N., and Levine, R. An iterative nonlinear Gaussianization algorithm for resampling dependent components. *Proc. 2nd International Workshop on Independent Component Analysis and Blind Signal Separation*, pp. 245–250, 2000.

Liu, H., Han, F., Yuan, M., Lafferty, J., and Wasserman, L. High dimensional semiparametric Gaussian copula graphical models. *The Annals of Statistics*, 40(4):34, 2012.

Lyu, S. and Simoncelli, E. P. Nonlinear extraction of independent components of natural images using radial Gaussianization. *Neural computation*, 21:1485–1519, 2009.

Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. In *NIPS*, 2017.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Ram, P. and Gray, A. G. Density estimation trees. In *KDD*, pp. 627–635, 2011.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014.

Tabak, E. G. and Turner, C. V. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013.

Tabak, E. G. and Vanden-Eijnden, E. Density estimation by dual ascent of the log-likelihood. *Commun. Math. Sci.*, 8(1):217–233, 2010.

# A. Tree Destructor Details

We give more details on how to compute the tree destructor transformation based on a density tree. Note that the tree destructor transformation is simply a piecewise independent linear transformation that depends on the leaf node of the sample.

### A.1. Base case: One dimension and one split

The base case of density tree transformation is only acting on one dimension while leaving all the others fixed. Thus, we can simply consider a tree in one dimension with only one split. We denote $[a, b]$ to be the domain of a node, $t \in [a, b]$ is the splitting threshold, and $p \in [0, 1]$ is the relative probability of the left child (the relative probability of the right child is just $(1 - p)$). Thus, the parameters of a node are simply $(a, b, t, p)$.

The transformation for the left $T_\ell(\cdot)$ and right $T_r(\cdot)$ of the split is as follows:

$$t_{\text{out}} \equiv (b - a)p + a \tag{9}$$

$$T_\ell(x) = \frac{t_{\text{out}} - a}{t - a}(x - a) + a \tag{10}$$

$$T_r(x) = \frac{b - t_{\text{out}}}{b - t}(x - t) + t_{\text{out}}\,, \tag{11}$$

where $t_{\text{out}}$ can be viewed as the "output threshold" (i.e. where the threshold in the output domain for this node should be to yield a uniform density). This yields the following scales $\alpha$ and shifts $\beta$:

$$\alpha_\ell = \frac{t_{\text{out}} - a}{t - a} \tag{12}$$

$$\beta_\ell = a - a\alpha_\ell \tag{13}$$

$$\alpha_r = \frac{b - t_{\text{out}}}{b - t} \tag{14}$$

$$\beta_r = t_{\text{out}} - t\alpha_r\,. \tag{15}$$

### A.2. Inductive Case

The full recursive details for a deeper tree is straightforward. We merely accumulate the linear transformation as we traverse the tree but only apply the transformation at the leaves. Composing any two linear operations can be simplified to a single linear operation:

$$T_1(T_2(x)) = \alpha_2(\alpha_1 x + \beta_1) + \beta_2 \tag{16}$$

$$\tilde{T}_{12}(x) = \alpha_2\alpha_1 x + \alpha_2\beta_1 + \beta_2 \tag{17}$$

$$= \tilde{\alpha}x + \tilde{\beta} \tag{18}$$

Thus accumulating transformations when traversing the tree only requires computing shift and scale values (as needed) for each node based on previous nodes and the current node.

# B. Enlarged Figures

See next pages for enlarged figures.

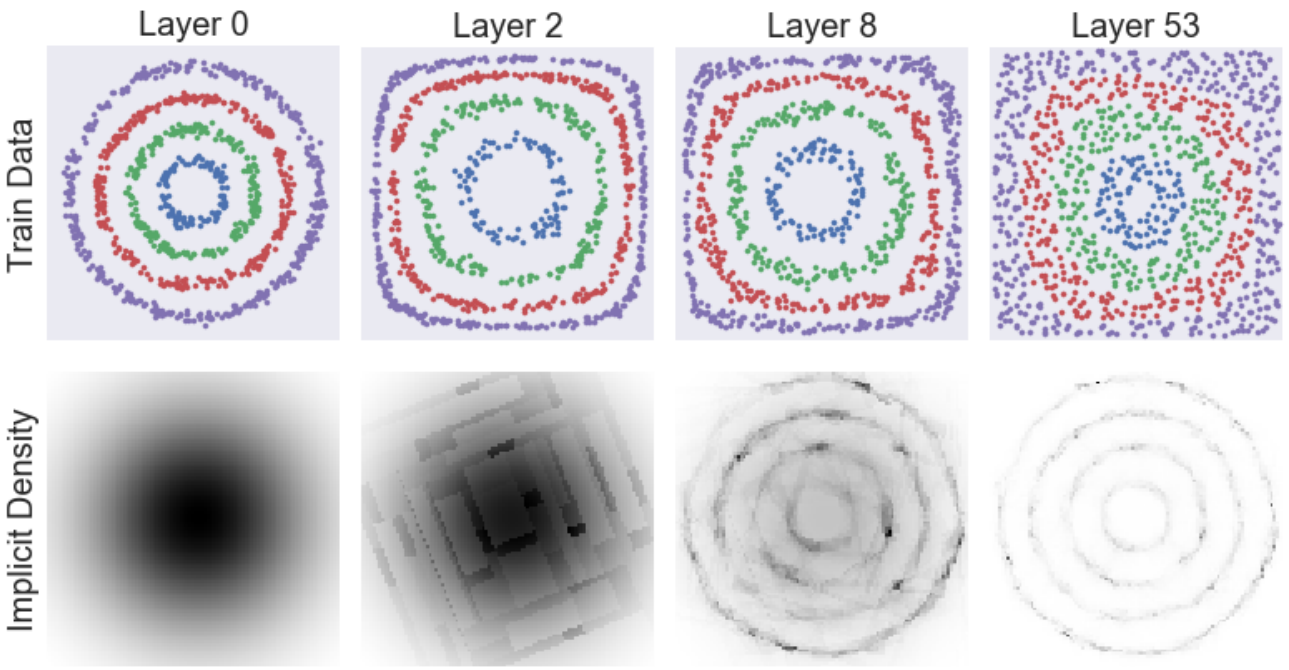


*Figure 5.* The transformed samples (top) and implicit density (bottom) at different layers of the DensityTree (100) model described in Sec. 4. While the initial layers do not provide a good estimate of the underlying density, after many layers, the underlying density begins to model the true underlying patterns.

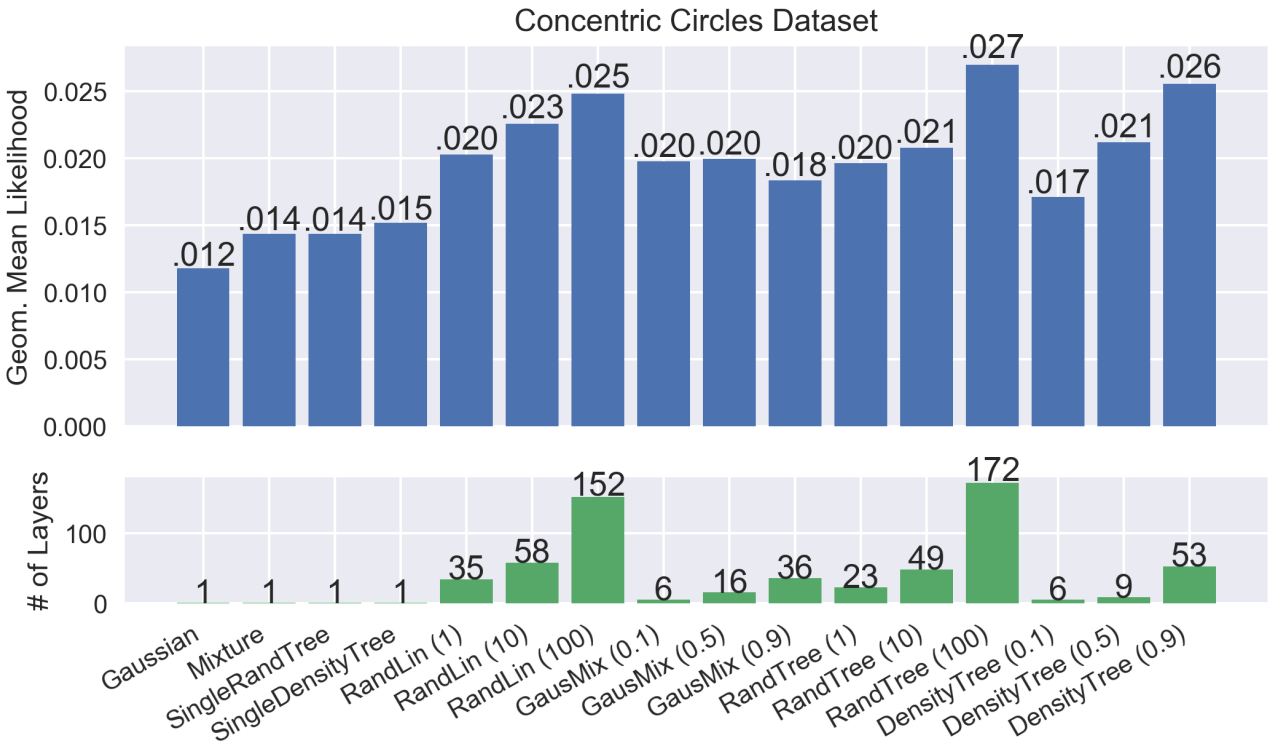


*Figure 6.* Deep models clearly outperform baseline models, and random trees seem to perform the best in terms of test likelihood. As expected, the selected number of layers increases as the amount of regularization on each layer increases.

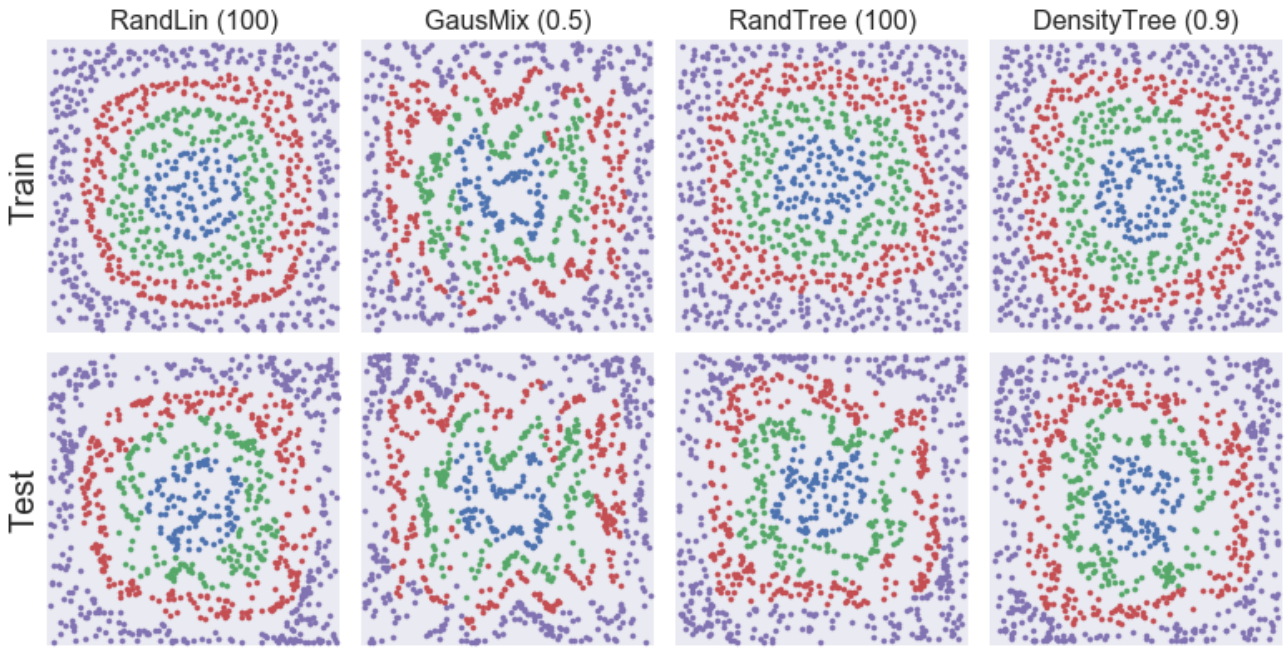


*Figure 7.* Each deep model transforms the data differently with DensityTree seeming to be the most regular. Notice that while the train data may be evenly spaced on the unit square, this does not mean the test data is also uniform. This is why highly regularized destructors are required to build usable deep models via a greedy algorithm.