A new perspective on probabilistic image modeling

Alexander Gepperth
Fulda University of Applied Sciences
Fulda, Germany
alexander.gepperth@cs.hs-fulda.de

***Abstract*—We present the Deep Convolutional Gaussian Mixture Model (DCGMM), a new probabilistic approach for image modeling capable of density estimation, sampling and tractable inference. DCGMM instances exhibit a CNN-like layered structure, in which the principal building blocks are convolutional Gaussian Mixture (cGMM) layers. A key innovation w.r.t. related models like sum-product networks (SPNs) and probabilistic circuits (PCs) is that each cGMM layer optimizes an independent loss function and therefore has an independent probabilistic interpretation. This modular approach permits intervening transformation layers to harness the full spectrum of (potentially non-invertible) mappings available to CNNs, e.g., max-pooling or (half-)convolutions. DCGMM sampling and inference are realized by a deep chain of hierarchical priors, where samples generated by each cGMM layer parameterize sampling in the next-lower cGMM layer. For sampling through non-invertible transformation layers, we introduce a new gradient-based sharpening technique that exploits redundancy (overlap) in, e.g., half-convolutions. The basic quantities forward-transported through a DCGMM instance are the posterior probabilities of cGMM layers, which ensures numerical stability and facilitates the selection of learning rates. DCGMMs can be trained end-to-end by SGD from random initial conditions, much like CNNs. We experimentally show that DCGMMs compare favorably to several recent PC and SPN models in terms of inference, classification and sampling, the latter particularly for challenging datasets such as SVHN. A public TF2 implementation is provided as well.**

***Index Terms*—Deep Learning, Gaussian Mixture Models, Deep Learning, Deep Convolutional Gaussian Mixture Models, Stochastic Gradient Descent**

I. Introduction

This conceptual work is in the context of probabilistic image modeling by a hierarchical extension of Gaussian Mixture Models (GMMs), which we term Deep Convolutional Gaussian Mixture Model (DCGMM). Main objectives are density estimation, image generation (sampling) and tractable inference (e.g., image correction or in-painting).

Many recent approaches (e.g., GANs, VAEs) excel in image generation by harnessing the full spectrum of CNN transformations, such as convolutions or pooling. An issues with these models is however the lack of density estimation and tractable inference capacity, i.e., explicitly expressing and exploiting the learned probability-under-the-model $p(\boldsymbol{x})$ of an image $\boldsymbol{x}$. In contrast, recent "deep" probabilistic approaches like sum-product-networks and probabilistic circuits [3], [15]–[17], [23] have been explicitly designed to perform these functions for high-dimensional visual problems. However, strong constraints must be satisfied at every hierarchy level to maintain a global probabilistic interpretation, which restricts the spectrum of available transformations. In particular, overlapping convolutions and pooling are excluded, with negative impact on, e.g., the capacity for realistic sampling. We aim at overcoming these limitations by constructing deep hierarchies of convolutional Gaussian Mixture Models (GMMs) in order to perform density estimation, realistic sampling and inference within a single model for large-scale, complex visual problems.

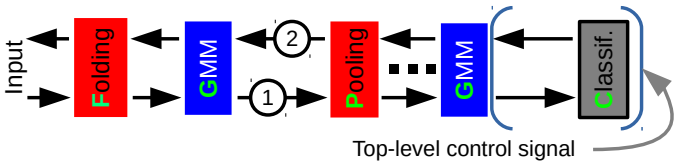

Fig. 1. High-level structure of a typical DCGMM instance with principal layer types. DCGMMs have two operation modes: ①the forward mode which estimates the probability of a presented data sample ②the backwards mode in which a top-level control signal is propagated in reverse direction to generate a sample according to the learned model. The whole architecture is trained end-to-end, with each trainable layer being independently optimized.

A. DCGMM: Model overview and salient points

DCGMM instances can operate in density estimation ("forward") and sampling ("backwards") mode, and are realized by a succession of convolutional Gaussian Mixture Model (cGMM) and non-trainable transformation layers, see Fig. 1.

The most prominent novel point in DCGMMs is the optimization of **independent loss functions** for each cGMM layer. This ensures a proper probabilistic interpretation of all cGMM layer outputs and removes the need for structural constraints as they are imposed in related probabilistic models (see Sec. I-B) which optimize a a single loss function for the whole hierarchy. Since DCGMM relaxes the assumption of a single loss function, arbitrary transformations may be performed between cGMM layers. In this article, we consider two **standard CNN operations**: half-convolution and pooling, which are non-invertible to different degrees.

In order to allow **sampling through non-invertible operations**, we exploit the fact that DCGMM convolutions are unconstrained and can thus be overlapping as in CNNs. We introduce a novel gradient ascent technique that exploits the redundancy thus created to compensate for the information loss due to non-invertible transformations.

The Gaussian Mixture Models used in DCGMMs are **inherently convolutional** in nature, see Fig. 2 and have a natural synergy with half-convolutions. A cGMM layer with

parameters π_k, $\mathbf{\Sigma}_k$ and $\boldsymbol{\mu}_k$ models the channel content at a certain *position* of a four-dimensional NHWC input tensor (see Fig. 2), where the cGMM parameters are *shared* between positions. The input tensor has usually been created by a half-convolution layer, which dumps the content of each overlapping *patch* of a NHWC tensor (mini-batch of images) into the channel content at that patch's position. This kind of parameter sharing between positions allows powerful models to be trained with relatively few trainable parameters in perfect analogy to CNN layers.

As an important design choice, we chose to **propagate cGMM posterior probabilities** (bounded and normalized) instead of log-probabilities. Although slightly more expensive, this choice provides numerical stability (no underflows), allows virtually problem-independent learning rates, and avoids complex gradient-clipping procedures.

DCGMM sampling realizes a **deep chain of hierarchical priors**, see [9]: Since a cGMM layer forward-propagates probability estimates for its own latent variables, their distribution is modeled by the next upstream cGMM layer. A sample drawn from the upstream cGMM thus represents a likely realization of these probabilities. As such, it naturally parameterizes the multi-nomial distribution that selects the Gaussian component to sample from. The generated sample, in turn, serves to parameterize sampling in a lower cGMM layer.

Due to these conceptually novel points, DCGMMs can not only perform density estimation and inference, but also generate realistic samples from **highly complex visual problems**, such as represented by the Fruits and SVHN benchmarks[1]. To the best of our knowledge, this has not been achieved before by any related work on probabilistic image modeling.

B. Related Work

GANs and VAEs and related models The currently most widely used models of image modeling and generation are GANs [2], [6], [12] and VAEs. GANs are capable of sampling photo-realistic images [18], but are unable to perform density estimation. Furthermore, GANs may suffer from what is termed *mode collapse*, which is hard to detect automatically due to the absence of a loss function [18]. VAEs show similar performance when it comes to sampling, in addition to minimizing a differentiable loss function. On the other hand, density estimation with VAEs is problematic as well, although the use of ELBO allows efficient outlier detection. Inference with VAEs is possible but complicated and potentially inefficient. Similar approaches, with similar strength in sampling but lack of density estimation, are realized by the PixelCNN architecture [14], GLOW [8] and their variants.

[1]CIFAR is not considered a valid benchmark for image modeling, since there are too few images per category and the categories themselves are too variable

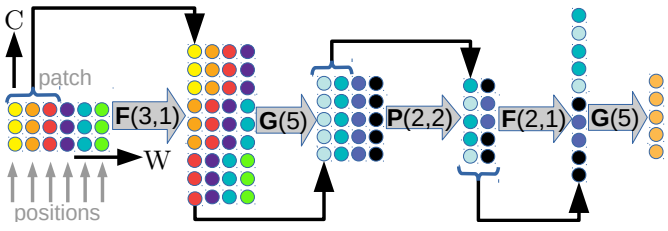

Fig. 2. A simple DCGMM instance applied to a NHWC tensor with N and W dimensions omitted. It contains max-pooling(P), half-convolution/folding(F) and cGMM (G) layers, see text for details. Lower cGMM layers analyze local image *patches* at different *positions*. The topmost cGMM layer is global in the sense that it has a single output position only, in which the whole image is described. Each cGMM layer provides a "fresh" probabilistic interpretation, indicated by uniform coloring for each position.

MFA MFA models [5], [11] are a generalization of GMMs which interpolate between GMMs with diagonal –vs– full covariance matrices. Although MFA models are not hierarchical, it is shown in [18] that they compare quite favorably to GANs when considering image generation.

Hierarchical MFA/GMM Two related approaches are described in [21], [22]. These approaches realize hierarchical MFA as modular decompositions of single "flat" MFAs, and thus possess a single loss function that is optimized by variants of Expectation-Maximization (EM). Closer to our approach is the proposal of [19], which proposes to train an MFA layer on the inferred latent variables of another, independent MFA instance. Transformations occurring in these models are global, i.e., affect *all* input variables. Local operations like convolutions or max-pooling are not used, and the dimensionality of samples treated in the experiments is low. A preliminary version of DCGMMs was described in [1].

Probabilistic circuits and sum-product networks Probabilistic circuits are deep directed graphs whose leaves compute probabilities from inputs via tractable parameterized distributions. These probabilities are further processed at weighted-sum- and product nodes, with the aim of obtaining a tractable distribution at the root node. This is ensured if some structural constraints are met, and the resulting PCs are often termed Sum-Product Networks (SPNs, see, [17]). In particular, SPNs allow efficient sampling, density estimation and tractable inference even if graphs are very deep.

Similarly to neural networks, finding a graph structure suitable for a particular learning task is challenging, especially if data dimensionality is high. Several interesting ideas for this have been put forward in [3], [15]–[17], [23]. An appealing approach is realized by RAT-SPNs [16] which construct random SPNs that are then combined by a single root node, at the price of additional hyper-parameters. The optimization of learning and inference in SPNs is addressed by Einsum Networks [15] where it is shown that successive sum and product nodes can be efficiently combined into an *Einsum operation*, which is particularly suited to GPU-friendly implementations.

SPNs suffer from the structural constraints required for representing valid, tractable distributions. These constraints exclude key CNN ingredients such as overlapping convolutions or max-pooling. Convolutions can be performed by SPNs in

certain special cases [3], [23], but general CNN-like convolutions remain inaccessible.

SPNs have been successfully be trained on large-scale, high-dimensional visual problems [3], [16], [16], [23] like the MNIST, FashionMNIST and SVHN benchmarks. Sampling and inference in SPNs are mostly demonstrated for the "Olivetti faces" benchmark [3], [17], [23] which is high-dimensional but contains only a few hundred samples. To the best of our knowledge, sampling has not been demonstrated for MNIST and FashionMNIST. In [15], training, sampling and inference is demonstrated based on the SVHN benchmark using a very simple SPN architecture and extensive data preprocessing. Generated samples are of good quality but not yet comparable to those produced by GANs or VAEs.

An interesting overview of the current research landscape in terms of hierarchical generative mixture models is given in [7]. All of these models are, in principle, capable of sampling and density estimation although the quality of sampling, and the data they can be trained on, vary considerably.

C. Objective, Contribution and Novelty

The objectives of this article are to introduce a deep GMM architecture which exploits the same principles that led to the performance explosion of CNNs. Novel points are:

- fundamentally new, modular approach for hierarchical mixture modeling based on independent mixture models
- fundamentally new approach to sampling as a deep chain of hierarchical priors
- use of arbitrary transformations like convolution and pooling in probabilistic image modeling
- realistic sampling for complex visual tasks despite non-invertible operations (pooling, convolutions)
- scalable and numerically robust end-to-end training by SGD from random initial conditions (no k-means initialization)
- experimental comparison to various models based on sum-product networks and probabilistic circuits

In addition, we provide a publicly available TensorFlow2 implementation.

II. Datasets

For the evaluation we use the following image datasets:

MNIST [10] is the common benchmark for computer vision systems and classification problems. It consists of $60\,000$ 28×28 gray scale images of handwritten digits (0-9).

FashionMNIST [24] consists of images of clothes in 10 categories and is structured like the MNIST dataset.

SVHN [13] contains color images of house numbers (0-9, resolution 32×32).

These datasets are not particularly challenging w.r.t. classification, but their dimensionality of 784 (MNIST, FashionMNIST) and 3072 (SVHN) is high, and the variability of especially the SVHN is considerable, which poses strong challenges for generative image modeling.

III. Methods: Deep convolutional Gaussian Mixture Models

In order to introduce DCGMMs as a possible hierarchical generalization of GMMs, we will first review facts about vanilla GMMs and introduce the basic notation, which will subsequently be generalized to a multi-layered structure.

A. Review of GMMs

GMMs are probabilistic models that aim to explain the observed data $X = \{\boldsymbol{x}_n\}$ by expressing their density as a weighted mixture of K Gaussian component densities $\mathcal{N}(\boldsymbol{x}_n; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \equiv \mathcal{N}_k(\boldsymbol{x}_n)$:

$$p(\boldsymbol{x}_n) = \sum_{k=1}^{K} \pi_k \mathcal{N}_k(\boldsymbol{x}_n). \tag{1}$$

The normalized component weights π_k represent another set of GMM parameters, which modulate the overall influence of each Gaussian density.

GMMs assume that each observed data sample $\{\boldsymbol{x}_n\}$ is drawn from one of the Gaussian component densities $\mathcal{N}_k$. The selection of this component density is assumed to depend on an unobservable *latent variable* $z_n \in \{1, \ldots, K\}$ which follows an unknown distribution. This is formalized for a GMM with K components by formulating the *complete-data likelihood* for a single data sample as:

$$\begin{aligned} p(\boldsymbol{x}_n, z_n) &= \pi_{z_n} \mathcal{N}_{z_n}(\boldsymbol{x}_n) \\ p(\boldsymbol{X}, \boldsymbol{z}) &= \prod_n p(\boldsymbol{x}_n, z_n) \end{aligned} \tag{2}$$

For a single sample, we can compute the *GMM posteriors* or *responsibilities* $\boldsymbol{\gamma}(\boldsymbol{x}_n)$:

$$\gamma_k(\boldsymbol{x}_n) = p(z_n{=}k|\boldsymbol{x}_n) = \frac{p(\boldsymbol{x}_n, z_n{=}k)}{\sum_j p(\boldsymbol{x}_n, z_n{=}j)} = \frac{\pi_k \mathcal{N}_k(\boldsymbol{x}_n)}{\sum_j \pi_j \mathcal{N}_j(\boldsymbol{x}_n)}, \tag{3}$$

which can be computed without reference to the latent variables. Marginalizing the unobservable latent variables out of Eq. (2) and taking the log, we obtain the *incomplete-data likelihood* $\mathcal{L}$:

$$\begin{aligned} \mathcal{L} = \log p(X) = \log \prod_n p(\boldsymbol{x}_n) = \log \prod_n \sum_k p(\boldsymbol{x}_n, z_n{=}k) = \\ = \log \prod_n \sum_k \pi_k \mathcal{N}_k(\boldsymbol{x}_n) = \sum_n \log \sum_k \pi_k \mathcal{N}_k(\boldsymbol{x}_n). \end{aligned} \tag{4}$$

The function $\mathcal{L}$ contains only observable quantities and is a suitable loss function for optimization.

At stationary points of Eq. (4), the component weights represent the average responsibilities: $\pi_k = \mathbb{E}_n \gamma_k(\boldsymbol{x}_n)$. For sampling, it is therefore common to first draw a latent variable sample $\hat{z}$ from a multinomial distribution $\mathcal{M}(\boldsymbol{\pi})$ parameterized by the weights $\boldsymbol{\pi}$, and then use the selected component density $\mathcal{N}_{\hat{z}}$ for generating the sample $\hat{\boldsymbol{x}}$:

$$\begin{aligned} \hat{z} &\sim \mathcal{M}(\boldsymbol{\pi}) \\ \hat{\boldsymbol{t}} &\sim \mathcal{N}_{\hat{z}}. \end{aligned} \tag{5}$$

Layer type	Notation	H	W	C
Folding	$F(f,\Delta)$	$1+\frac{H'-f}{\Delta}$	$1+\frac{W'-f}{\Delta}$	f^2C'
Max-Pooling	$P(f,\Delta)$	$1+\frac{H'-f}{\Delta}$	$1+\frac{W'-f}{\Delta}$	C'
Classification	$C(S)$	1	1	S
Conv. GMM	$G(K)$	H'	W'	K

TABLE I

OVERVIEW OF DCGMM LAYER TYPES. THE THREE RIGHTMOST COLUMNS INDICATE THE SHAPE OF ACTIVITIES IN FORWARD MODE IF THE LAYER RECEIVES AN INPUT OF DIMENSIONS H',W',C'. FOR FOLDING AND MAX-POOLING LAYERS, PARAMETERS ARE THE KERNEL SIZE f AND THE STRIDE Δ. FOR CGMM LAYERS, THE PRINCIPAL PARAMETER IS THE NUMBER OF COMPONENTS K.

B. DCGMM basics

The basic data format in a DCGMM instance (see Fig. 1) are four-dimensional NHWC tensors. For conciseness, we will omit the batch dimension (N) from all formulas. We will denote the dimensions of the current layer L as H,W,C and those of the preceding layer $L-1$ as H',W',C'. Lower-case indices are used similarly, and are sometimes grouped into tuples $\vec{m}=[h,w,c]^T$ or $\vec{m}'=[h',w',c']^T$ for brevity.

Contrary to DNNs, DCGMMs have two operational modes, see Fig. 1: *forward* for density estimation, and *backwards* for sampling. In forward mode, each DCGMM layer with index $L\geq 1$ receives input from the preceding layer $L-1$, and generates *activities* $\boldsymbol{A}^{(L)} \in \mathbb{R}^{\mathrm{H,W,C}}$ which serve as inputs to the subsequent layer $L+1$. As per the usual convention, $L=0$ denotes the data samples themselves. In backwards mode, each layer L receives a *control signal* $\boldsymbol{T}^{(L)} \in \mathbb{R}^{\mathrm{H,W,C}}$from layer $L+1$ and produces another control signal $\boldsymbol{T}^{(L-1)} \in \mathbb{R}^{\mathrm{H',W',C'}}$ for layer $L-1$. We define three DCGMM layer types: *folding* layers implementing half-convolutions, *max-pooling layers* (analogous to CNNs, but including backwards mode) and *convolutional GMM layers*. Characteristics, notation and principal parameters of all layer types are given in Tab. I.

In the following text, we will discuss how the different DCGMM layer types implement computation of loss functions (where applicable), activities and control signals.

C. Convolutional Gaussian Mixture Layer

cGMM layers are realized by GMMs, slightly modified to a convolutional formulation. As with any GMM (see Sec. III-A), this layer type is defined by weights $\pi_k^{(L)}$, centroids $\boldsymbol{\mu}_k^{(L)}$ and covariance matrices $\boldsymbol{P}_k^{(L)}$.

Forward mode In this mode, activities $\boldsymbol{A}^{(L)}$ of the layer are computed as a function of preceding layer activities $\boldsymbol{A}^{(L-1)}$. Specifically, activities are realized by GMM posteriors, see Sec. III-A. Since the layer is convolutional, these posteriors are computed from the channel content $A_{hw,:}^{(L-1)}$ at every position h,w in the tensor of input activities $\boldsymbol{A}^{(L-1)}$:

$$\begin{aligned} P_{hw,k}^{(L)}\big(\boldsymbol{A}^{(L-1)}\big) &= p\big(A_{hw,:}^{(L-1)}\big) \\ A_{hw,k}^{(L)}\big(\boldsymbol{P}^{(L)}\big) &= \gamma_k\big(P_{hw,:}^{(L)}\big) \end{aligned} \tag{6}$$

Backwards mode In this mode, the cGMM layer generates the control signal $\boldsymbol{T}^{(L-1)}$ by sampling with an hierarchical prior defined by the up-stream control signal $\boldsymbol{T}^{(L)}$ as detailed in Sec. III-A. Sampling is performed for each position h,w of the up-stream control signal:

$$\begin{aligned} \hat{Z}_{hw}^{(L-1)} &\sim \mathcal{M}\big(T_{hw,:}^{(L)}\big) \\ T_{hw,:}^{(L-1)} &\sim \mathcal{N}_{\hat{Z}_{hw}^{(L-1)}} \end{aligned} \tag{7}$$

In case there is no control signal (layer is topmost DCGMM layer), the $\hat{Z}_{hw}^{(L-1)}$ are drawn from an uniform distribution.

Loss function For efficient and numerically stable training, we use an approximation to the GMM log-likelihood, which we term the *max-component approximation*. This approximation was validated in [4] and is analogous to Expectation-Maximization training with hard assignments, see, e.g., [22]. As stated in Sec. I-A, each cGMM layer has a loss function that is computed and optimized independently of other layers. For a single sample, it reads:

$$\mathcal{L}^{(L)} = \mathbb{E}_{h,w} \log \max_k P_{hw,k}^{(L)} \tag{8}$$

An essential point about cGMM layers is that the activities $\boldsymbol{A}^{(L)}$ and control signals $\boldsymbol{T}^{(L-1)}$, are computed using a **single** set of centroids, weights and covariance matrices at every position. Thus, these quantities are **shared** in the same way CNN filters are shared across an image. In this way, large images can be described while requiring relatively few trainable parameters. If memory consumption is not an issue, independent parameters can be used for each position as well.

D. Max-Pooling Layer

This layer type performs a complex, deterministic transformation in both modes, which would normally render the represented density intractable. It is the task of subsequent cGMM layers to re-instate a tractable probabilistic interpretation. Pooling is governed by a *kernel size* f and a *step size* Δ, see Fig. 2, where we usually assume non-overlapping pooling: $f=\Delta$. To formalize this, we assign to every position $\vec{m}$ a *receptive field*:

$$\nu(\vec{m}) = \{\vec{m}' : h' \in [h\Delta, h\Delta+f[, w' \in [w\Delta, w\Delta+f[, c'=c\} \tag{9}$$

In forward mode, this layer type behaves exactly like a CNN max-pooling layer, see also Tab. I and Fig. 2. The relation is $A_{\vec{m}}^{(L)} = \max_{\vec{m}' \in \nu(\vec{m})} A_{\vec{m}'}^{(L-1)}$. In backwards mode, the pooling layer aims at reconstructing a tensor that would have resulted in the provided control signal. Numerous schemes are possible due to the non-invertibility of the mapping. We choose to perform *upsampling* here: $T_{\vec{m}' \in \nu(\vec{m})}^{(L-1)} = T_{\vec{m}}^{(L)}$. The non-uniqueness of the backwards mapping must be counteracted by additional measures such as sharpening, see Sec. IV-C.

E. Folding Layer

This layer type performs a deterministic non-invertible mapping. Again, the represented density is rendered intractable by this until the next cGMM layer.

Forward mode In this mode, folding layers perform a half-convolution on their inputs, governed by a *kernel size* f and

a *step size* Δ, see also Tab. I and Fig. 2. The forward mode transformation reads

$$\kappa(\vec{m}) = \begin{pmatrix} h\Delta + c//(fC') \\ w\Delta + (c//C')\%f \\ c\%C' \end{pmatrix} \tag{10}$$

$$A^{(L)}_{\vec{m}} = A^{(L-1)}_{\kappa(\vec{m})}, \tag{11}$$

where // and % represent integer and modulo division.

Backwards mode Here, we invert $\kappa(\vec{m})$ to obtain a control signal. Due to the one-to-many nature of the mapping $\kappa(\vec{m})$, this can be done in J different ways, which we choose to average over:

$$T^{(L-1)}_{\vec{m}'} = \frac{1}{J} \sum_{\substack{\vec{m} \\ \kappa(\vec{m})=\vec{m}'}} T^{(L)}_{\vec{m}}. \tag{12}$$

F. Classification Layer

This layer type implements a simple linear classifier for S classes, and is usually situated at the top of a DCGMM instance. Trainable parameters are the weight matrix $\boldsymbol{W}$ and the bias vector $\boldsymbol{b}$. In forward mode, it generates per-class probabilities as $p^{(L)}_s(\boldsymbol{x}) = S_s(\boldsymbol{x}W + \boldsymbol{b})$, where S_s represents component s of the softmax function, and $\boldsymbol{x}$=flatten$\left(\boldsymbol{A}^{(L-1)}\right)$. n backwards mode, it performs an approximate inversion of this operation, transforming the top-level control signal $\boldsymbol{y} \in \mathbb{R}^S \equiv \boldsymbol{T}^{(L)}$ into a control signal for layer $L-1$:

$$\boldsymbol{T}^{(L-1)} = \text{reshape}_{H'W'C'} W^T \left(\log(\boldsymbol{y}) - \boldsymbol{b} + \text{const.}\right) \tag{13}$$

The control signal $\boldsymbol{y}$ must contain a one-hot encoding of the class that should be generated. The constant that arises due to the ambiguity in inversing the softmax must be chosen such that the control signal is positive, as it must be if it is to represent GMM posteriors. Just as cGMM layers, classification layers optimize an independent loss function given by standard cross-entropy (which requires target values).

IV. Architecture-Level DCGMM Functionalities

A. End-to-end DCGMM training

A defining characteristic of DCGMMs is the fact that there is no "global" loss function. Instead, each trainable layer independently optimizes its own loss function, and this would usually be performed in a greedy layer-wise fashion as, e.g., the original denoising autoencoder models. In order to speed up learning, we propose a joint training scheme, where all layers are optimized at the same time. To avoid convergence problems, we start adapting the trainable layer L at time $\delta^{(L)}$, which increases with the position in the hierarchy. Thus, lower layers achieve some convergence before higher layers start their adaptation process, which works extremely well in practice for very small delays $\delta^{(L)}$.

Training cGMM layers is performed by SGD from random initial conditions as detailed in [4]. As explained in [4], SGD parameters are very robust and are kept constant throughout all experiments in this article.

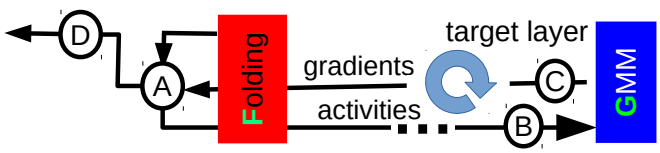

Fig. 3. The gradient-based sharpening procedure, see text for details. Ⓐ Control signal sampled from folding layer Ⓑ forward propagation of control signal to chosen target GMM layer Ⓒ back-propagated gradients applied for modifying control signal Ⓓ final control signal passed to layer $L-1$.

B. Density estimation and outlier detection

This is the principal function performed in forward mode, see Fig. 1. In contrast to, e.g., deep MFA or SPN instances, see Sec. I-B where sample probability is computed from the root node activation, any cGMM layer L in a DCGMM instance expresses sample probability by its log-likelihood $\mathcal{L}^{(L)}$. Whereas cGMM layers low in the hierarchy usually model the density of small image patches, higher cGMM layers capture more of the global image structure. We therefore assume that the highest cGMM layers are the best candidates for density estimation. We will evaluate this by performing outlier detection in Sec. VI-B.

C. Sampling and sharpening

Sampling is the principal function performed in backwards mode, see Fig. 1. Triggered by a user-defined or uniform top-level control signal, each layer L operates in backwards mode generating controls signals $\boldsymbol{T}^{(L-1)}$ until the sampling result is read out for L=0. Just as density estimation in forward mode, sampling in backwards mode can be efficiently conducted in mini-batches. As detailed in [1], control signals can be thresholded and renormalized to contain only the S highest values, which controls the variability of sampling.

The max-pooling and folding layer types perform forward-mode transformations that are not invertible. In backwards mode, this means that they can generate control signals that differ from the true data statistics. To rectify this, we first observe that up-stream cGMM layers "know" the correct statistics of their inputs through training. Additionally, folding layer outputs contain redundancies since their receptive fields usually overlap, which can be exploited.

We therefore propose a generic statistics-correcting procedure applied at each folding layer L, see Fig. 3: we first select a *target layer* $L'>L$, which is normally chosen as the next-highest cGMM layer in the DCGMM instance. First, we operate the folding layer in backwards mode to obtain the initial control signal $\boldsymbol{T}^{(L-1)}(i{=}0)$. Then, we perform gradient ascent for $i < I$ iterations, maximizing the target layer loss $\mathcal{L}^{(L')}(i)$ obtained by forward-propagating $\boldsymbol{T}^{(L-1)}(i{<}I)$. In this fashion, we obtain a statistics-corrected control signal $\boldsymbol{T}^{(L-1)} \equiv \boldsymbol{T}^{(L-1)}(i{=}I)$.

D. Conditional sampling

By training a top-level classification layer (see Fig. 1) and providing an appropriate control signal (see Sec. III-F),

DCGMMs can be enabled to selectively sample from certain classes only. This is very memory efficient since it removes the need to duplicate structures for every class that should be sampled from, as it is done in [16], [23].

E. Inference for variant generation and in-painting

Both functionalities are cases of tractable inference, which is built-in since every cGMM layer has this ability. For variant generation, an image sample is presented and the DCGMM instance must generate "similar" ones. In in-painting, we present a partially occluded image and ask the DCGMM instance to complete the missing part.

The procedure in both cases is to perform a forward pass up to a certain cGMM layer X, and then perform a backwards pass with $\boldsymbol{T}^{(X)}=\mathbf{A}^{(X)}$ downwards from layer X. For in-painting, X must correspond to a cGMM layer that depends on all image variables, usually the topmost one, and the sampling result for the unknown image part is used to complete the input sample. For variant generation, X can be an intermediate cGMM layer as well, which allows to control the global variability of the generated variants.

V. Methods: Models and Parameters

DCGMM We define 7 distinct DCGMM instances which are summarized in Sec. IX. The instances differ in depth and by their use of stepped folding –vs– max-pooling. The principal hyper-parameters are the number of components in cGMM layers, which are always chosen such that a batch size of 100 remain feasible. In several experiments, we use (or do not use) parameter sharing between different input positions, for specific layers. Learning rates and other SGD-related parameters are kept exactly as described in [1], [4]. Since these parameters do not appear to be task-dependent, we do not vary them in the presented experiments. The training delay for cGMM layer L is set to $\delta^{(L)}=0.1(L)$, where $L\in\{1,2,\dots\}$ indicates the number of cGMM layers below the current one.

RAT-SPN We implemented RAT-SPNs as described in the original publication [16] using the implementation proposed in [16]. As in [15], for every experiment we vary the following parameters: number of distributions per leaf region/number of sums per sum node $I, S \in \{5, 20, 40\}$, split depth $D \in \{1, 5, 9\}$ and number of repetitions $R \in \{10, 25, 40\}$. To speed up training, we employ binomial leaf distributions. Training is conducted for 25 epochs. Variances are constrained as in [16].

Deep generalized convolutional SPNs (DGCSPNs) We implemented the convolutional architecture for generative experiments from the original publication [23] using *libspn-keras*. The number of sums per sum layer is varied as $S \in \{16, 32, 64\}$. Training is conducted for 15 epochs. Accumulators are initialized by a Dirichlet distribution with $\alpha = 0.1$, the number of normal leaf distributions is set to 4, and centroids are initialized from training data as described in [23], where it is also suggested that variances be kept constant at 1.0 for all normal leaves.

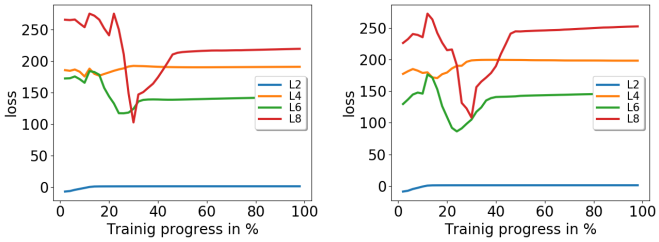

Fig. 4. Test losses for all 4 cGMM layers in the deep DCGMM-F instance for MNIST(left) and FashionMNIST(right). Please note that individual cGMM layers' losses have different ranges, and that they are maximized (not minimized) by DCGMM training!

Poon-Domingos SPN architecture As an SPN baseline, we use PD-SPN, a very simple instance of the Poon-Domingos architecture as described in [16] using the implementation proposed in [15]. As in [15], we use $\Delta = \{8\}$ for SVHN and $\Delta = \{7\}$ for MNIST, along the horizontal dimension only. We vary the number of distributions per leaf region/number of sums per sum node: $I, S \in \{5, 20, 40\}$.

VI. Experiments

Compared models are DCGMM, RAT-SPN, PD-SPN and DGCSPN, using parameter (ranges) as given in Sec. V. All experiments were performed on a cluster of 50 off-the-shelf PCs using nVidia GeForce RTX 2060 GPUs, and our own TensorFlow-based implementation of DCGMMs. The used libraries are referenced in Sec. XIV. Generally, we repeat each experiment 5 times with same parameters but different initializations. When grid-searching for feasible parameters, we use the averaged performance measures to identify the best settings. Where feasible, we report mean and standard deviations for experiments.

A. DCGMM training dynamics

This experiment is conducted to demonstrate the basic feasibility of DCGMM training, despite the fact that we optimize a separate loss function for each cGMM layer. We train DCGMM architecture F, see Sec. IV, on the full MNIST and FashionMNIST dataset and record the test losses for each of the cGMM layers over time. cGMM Parameters are clamped for a certain percentage of training time in different cGMM layers, see Sec. V. As we can observe in Fig. 4, each cGMM layer achieves convergence after a certain time, starting with the lowest one and proceeding to the highest one which converges last. While a layer's parameters are clamped, its loss fluctuates, since it is impacted by parameter adaptation in lower layers. We can observe the un-clamping of parameters by the "dips" in each layer's loss, which occurs in a strictly sequential fashion following parameter un-clamping of each layer. The fact that losses are *higher* initially is explained by the fact that these are test losses.

ID	MNIST	FashionMNIST	ID	MNIST	FashionMNIST
A	**95.4**±0.2	62.2±0.3	E	92.8±0.4	**74.9**±0.2
B	93.3±0.5	72.2±0.2	F	84.1±0.4	68.1±0.1
C	94.2±0.2	74.3±0.1	G	77.4±0.3	62.7±0.1
D	92.5±0.3	68.1±0.1	-	-	-

TABLE II

OVERVIEW OF OUTLIER DETECTION CAPACITY OF VARIOUS DCGMM INSTANCES, SEE SEC. IX, MEASURED BY THE MEAN AUC (IN %).

ID	MNIST AUC in %	FMNIST AUC in %
DCGMM-E	**92.8**± 0.4	**74.9** ± 0.2
RAT-SPN	91.8 ± 0.7	39.2 ± 0.4
DGCSPN	90.6 ± 0.7	57.1 ± 0.9
PD-SPN	91.2± 0.6	48.5 ± 1.7

TABLE III

OUTLIER DETECTION PERFORMANCE FOR DCGMM-E AND SPN MODELS FOR MNIST AND FASHIONMNIST, QUANTIFIED BY THE AUC MEASURE.

ID	parameters	MNIST acc.	FMNIST acc.
DCGMM-A	38.416	98.2± 0.1	80.3± 2.6
DCGMM-B	293.657	98.7± 0.05	83.6± 1.7
DCGMM-E	40.850	99.1± 0.2	**94.5** ± 1.7
DCGMM-F	50.850	**99.9**± 0.07	89.0 ± 2.5
RAT-SPN	1.187.840	99.2± 0.1	85.4 ± 1.9
DGCSPN	7.560.576	98.0± 0.1	80.4± 6.0
PD-SPN	515.683	97.2± 0.2	81.2± 2.4

TABLE IV

SAMPLE GENERATION CAPACITY FOR DCGMM AND SPN MODELS AS MEASURED BY A CNN CLASSIFIER ON GENERATED SAMPLES, SEE TEXT FOR DETAILS. WE OBSERVE THAT ALL MODELS PERFORM VERY SIMILARLY ON MNIST, WHEREAS DCGMM-E OUTPERFORMS SPNS FOR FASHIONMNIST. AVERAGES AND STANDARD DEVIATIONS ARE TAKEN OVER 5 INDEPENDENT TRAINING RUNS OF THE CNN CLASSIFIER.

B. DCGMM outlier detection experiments

The goal of this experiment is to assess the outlier detection capabilities of DCGMMs. In particular, we investigate which DCGMM architectures are most suited for this purpose, and which cGMM layer the outlier detection should be based on. We construct outlier detection problems from classes 1–9 (inliers) –vs– 0 (outliers) of the MNIST and FashionMNIST datasets. Test losses on inlier and outlier classes are recorded for all layers in a DCGMM instance after training on the inliers. Outlier detection is evaluated separately based on each cGMM layer's loss. By varying the threshold for outlier detection (see Sec. IV-B), we obtain ROC-like outlier detection plots, see Sec. XI. The area-under-the-curve (AUC) is used as a quality measure. Results are summarized in Tab. III. We find that it is always the highest cGMM layers which are most suited for outlier detection, and that the depth of a DCGMM instance tends to increase its outlier detection capacity. The results with different outlier classes are comparable.

C. Outlier detection: model comparison

Here, we compare the outlier detection capacity of DCGMM-E, the best DCGMM instance from Sec. VI-B, to various RAT-SPN, PD-SPN and DGCSPN instances, using the same procedures. To find the best parameters for each SPN type, a grid search is conducted over parameters ranges as stated in Sec. V, repeating each experiment 5 times with identical parameters. Best outlier detection capacities, measured as in Sec. VI-B, are reported in Tab. III. We observe a clear edge for DCGMM-E, in particular for FashionMNIST.

D. Demonstrations of DCGMM sampling and sharpening

To demonstrate how the sharpening procedure of Sec. IV-C improves sampling through non-invertible operations, notably max-pooling, we train DCGMM instances (B and D, see Sec. IV) which perform max-pooling to sample from MNIST and FashionMNIST. Then, we compare sampling results with and without sharpening, see Fig. 5. When sharpening is used, target layers are always the next-highest cGMM layers, and gradient ascent is performed for $I = 300$ iterations using a learning rate of 1.0. Fig. 5 shows that strong blurring effects occur without sharpening, due to ambiguities in inverting max-pooling layers. In contrast, sharpening removes these ambiguities while maintaining diverse samples. This works best for DCGMM-B with a single max-pooling layer, whereas more max-pooling steps seem to destroy too much information, leading to rather frayed-looking samples. We conclude that "softer" strategies than max-pooling may be required for sampling with DCGMMs. The corresponding figures for FashionMNIST are given in Sec. XI and corroborate this view.

E. Sampling: visual model comparison

In this experiment, we perform a visual comparison of samples generated from DCGMM and the SPN models described in Sec. V. SPN models are used in the configurations given in the literature (see Sec. V) and the best parameters found by grid-search in Sec. VI-C. Since the configurations in the literature are tailored to certain datasets, we restrict this investigation to the MNIST and FashionMNIST datasets. For DCGMM, we selected DCGMM-F, see Sec. IX, an instance without max-pooling, since this generally leads to more realistic samples, see Sec. VI-D.

Typical samples generated by these instances are shown in Fig. 6. We observe that the DCGMM samples generally look smoother and more realistic than SPN-generated ones. Instances from other DCGMM instances, and the corresponding FashionMNIST samples, are shown in Sec. XII.

F. Sampling: quantitative model comparison

Since visual appearance can be deceptive, this experiment aims to provide a quantitative comparison of the sampling capacity of DCGMMs and SPNs, relying on MNIST and FashionMNIST. We first train a CNN classifier on both datasets to deliver state-of-the-art accuracy for CNNs (details in appendix Sec. X). DCGMMs and SPNs are separately trained on each class in both datasets, and asked to generate 1000 samples from all 10 classes. The quality measure is the CNN's classification accuracy on the generated samples. Results are shown in Tab. IV. We observe that DCGMM sampling seems to produce samples that more accurately match the real data than SPN-based models.

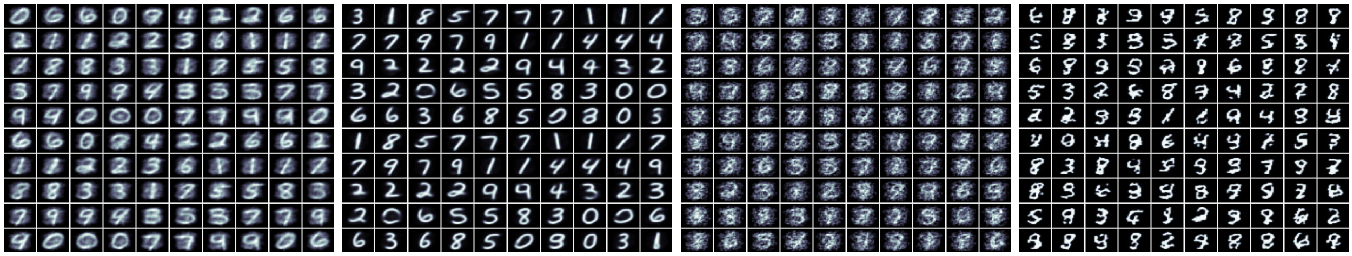

Fig. 5. Effects of sharpening on sampling quality. Sampling results from left to right: DCGMM-B (no sharpening), DCGMM-B (sharpening), DCGMM-D (no sharpening), DCGMM-D (sharpening). The most beneficial effect of sharpening is observed for the shallow DCGMM-B instance.

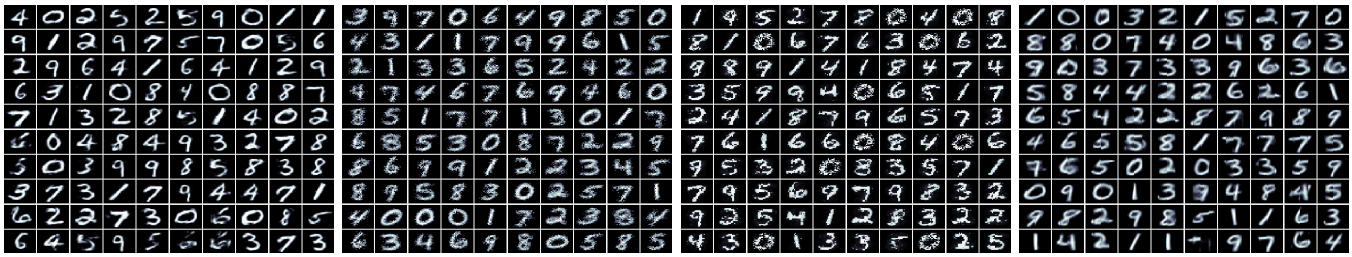

Fig. 6. Visual comparison of samples between DCGMM-E and several SPN models. Upper row, left to right: DCGMM-F, RAT-SPN, PD-SPN, DGCSPN.

Fig. 7. Sampling on SVHN. Left to right: a) SVHN samples b) VAE c) DCGMM-A d) DCGMM-B. Please note similar performance between b) and d).

G. Sampling for large-scale visual problems: SVHN

Here, we demonstrate that DCGMMs can be trained on large-scale, complex and high-dimensional visual problems and generate high-quality samples when compared to SPNs. To our knowledge, the only attempt to train SPNs on SVHN is described in [15], where a simplified instance of the Poon-Domingos (PD) architecture is trained on each of 500 clusters obtained by k-means treatment of the SVHN data. DCGMM instances A and B (the latter with the lowest cGMM layer weight-sharing mode, see Sec. III-C) are separately trained on each SVHN class, and then used for sampling. When comparing DCGMM samples to each other and to the PD-SPN samples shown in [15], we observe that the deeper DCGMM-B instance generates more diverse images, and that DCGMM samples do not exhibit the "stripe" artifacts present in [15], which are due to non-overlapping scopes in the PD architecture. DCGMM-B samples even compare favorably to samples generated by a VAE, see Sec. XI.

H. Inference: variant generation and in-painting

We perform inpainting as a special case of inference (see Sec. IV-E) on MNIST samples from which the right half was erased. In-painting has the same complexity as sampling, i.e., linear in the number of cGMM layers. Results for the shallow DCGMM-A and the deep DCGMM-F instance are shown in Fig. 8. We observe that completion fits the original image better with a deep DCGMM instance. This is natural since DCGMM-A is essential a vanilla GMM and can just replicate its limited set of components, with added Gaussian noise. A deep DCGMM instance, in contrast, exhibits much greater variability since each cGMM layer adds (guided) randomness in the choice of the component to sample from.

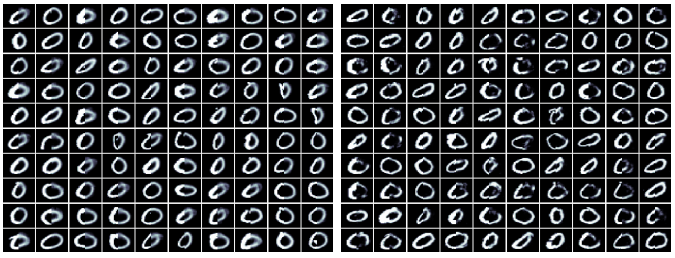

Fig. 8. Examples of in-painting the erased right half of an image with different DCGMM instances. Left: DCGMM-A, right: DCGMM-F.

model	MNIST acc. in %	FMNIST acc. in %
DCGMM-B	98.0 ± 0.1	89.6 ± 0.2
RAT-SPN	98.19	89.52
DGCSPN	98.66	90.74

TABLE V
CLASSIFICATION ACCURACIES IN % OBTAINED FOR MNIST AND FASHIONMNIST. FOR RAT-SPNS AND DGCSPNS, MEAN ACCURACY OVER 5 RUNS IS TAKEN FROM [23].

I. Generative-discriminative learning

An advantage of optimizing independent loss functions for each layer is the fact that discriminative training of a top-level classification layer does not influence unsupervised training in lower layers, since no back-propagation is applied. Thus, a DCGMM trained to classify MNIST digits will still be able to sample, in contrast to an SPN that optimizes a single cross-entropy loss. DCGMM classification accuracies are comparable if slightly inferior to those obtained using DGCSPN and RAT-SPN from the literature ([23]), see Tab. V. To boost DCGMM classification performance, we generally find it advantageous to use activities of the 2 highest cGMM layers as input to the classifier layer, not only the highest one.

VII. DISCUSSION AND CONCLUSION

Concerning DCGMMs, the experiments described in this article allow the following conclusions:

Sample probability This quantity is best expressed by the topmost cGMM layer of a DCGMM instance, see Sec. VI-B. A reasonable condition is that all image variables lie in the scope of this layer. This is an important results since DCGMMs optimize several loss functions, each of which provides a different (local) model of the data distribution.

Max-pooling We find that this operation is feasible in DCGMMs due to the sharpening procedure (see Sec. VI-D), and produces good results for outlier detection (see Sec. VI-B). However, in deep DCGMM instances, the information loss arising from max-pooling is too severe for good sampling performance. Here, stepped, overlapping half-convolutions are found to be a better choice.

Parameter sharing is very effective for reducing the number of model parameters especially in lower layers, and does not seem to impair sampling performance (see Sec. VI-G). This is probably because the local "visual alphabet" (see Sec. XIII) is nearly position-invariant in lower layers and thus can be shared between positions with little loss.

Parameter stability All experiments, regardless of dataset and classes, were conducted with the same set of DCGMM hyper-parameters, in particular learning rates, with no adverse effects. The only exception was the number of cGMM components K, which obeys a strict "the more the better" dependency and is thus easy to select. We see this parameter stability as an effect of propagating bounded and normalized cGMM posterior probabilities instead of unbounded and unnormalized log probabilities as it is done in SPNs.

The principal related work concerning DCGMMs are sum-product networks (SPNs) and Probabilistic Circuits (PCs) [15]–[17], [23]. With respect to these works, our experiments make the following points:

Realistic sampling Although the samples presented in, e.g., Sec. VI-D are not as realistic as those produced by, e.g., GANs, they have a clear edge w.r.t. samples produced by SPNs. This shows that DCGMMs can capture the overall probability distribution of image data quite well, and that linking independent probabilistic descriptions (the cGMM layers) by a deep chain of hierarchical priors is feasible.

Large-scale visual problems We showed in Sec. VI-G that simple DCGMMs can be trained on SVHN to generate samples of higher visual quality than shown in [15] for SPNs. One may speculate that the fact of having multiple *independent* probabilistic models of the data leads to greater expressive power, while requiring far fewer parameters.

Classification In DCGMMs, classification is achieved by adding a linear classifier layer at the top of a DCGMM instance. Due to the modular construction of DCGMMs, the classifier layer is optimized independently from the cGMM layers, which permits competitive classification(see Sec. VI-I) and sampling within the same instance. As reported in [23], RAT-SPNs and DGC-SPNs can perform competitive classification when trained using cross-entropy. However, since such training affects the whole network, the generative capabilities like sampling and inference are lost in this case.

Implementation aspects It is difficult to compare SPNs and DCGMMs in terms of learning and inference speed since it is not clear what constitutes equivalent models. Due to parameter sharing in DCGMMs, they generally have significantly less parameters (see Tab. IV). For models of roughly equal outlier detection performance, as those compared in Sec. VI-C, we find that an epoch of DCGMM training is performed roughly 2 times faster, depending on SPN implementation. On the other hand, SPN training generally converges much faster since EM is used instead of SGD. As a consequence, SPNs training is generally faster. The picture changes when taking into account that online EM training quickly diverges if parameters are not tuned to a particular task. Here, the parameter stability of DCGMMs saw to it that only a single learning rate could be used for all experiments.

VIII. REPRODUCIBILITY STATEMENT

All experiments in this paper are implemented in Python, based solely on publicly available software packages. No assumptions about a certain operating system or hardware configuration are made, although experiments will be accelerated significantly by GPU support. The software packages (and how to obtain them) used for SPN experiments are listed in Sec. XIV. We provide our TensorFlow2-based implementation of DCGMM as an anonymous repository, as indicated in Sec. XIV. Instructions how to run key experiments of this article can be found there as well.

REFERENCES

[1] anonymous citation.

[2] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[3] C. J. Butz, J. S. Oliveira, A. E. dos Santos, and A. L. Teixeira. Deep convolutional sum-product networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3248–3255, 2019.

[4] A. Gepperth and B. Pfülb. Gradient-based training of gaussian mixture models for high-dimensional streaming data. *Neural Processing Letters*, 2021.

[5] Z. Ghahramani and G. E. Hinton. The EM Algorithm for Mixtures of Factor Analyzers. *Compute*, pages 1–8, 1997.

[6] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 3(January):2672–2680, 2014.

[7] P. Jaini, P. Poupart, and Y. Yu. Deep homogeneous mixture models: Representation, separation, and approximation. In *NeurIPS*, pages 7136–7145, 2018.

[8] D. P. Kingma and P. Dhariwal. Glow: generative flow with invertible $1\times$ 1 convolutions. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 10236–10245, 2018.

[9] A. Klushyn, N. Chen, R. Kurle, B. Cseke, and P. van der Smagt. Learning hierarchical priors in vaes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998.

[11] G. McLachlan and D. Peel. Mixtures of Factor Analyzers. pages 238–256, jan 2005.

[12] M. Mirza and S. Osindero. Conditional Generative Adversarial Nets. pages 1–7, 2014.

[13] Y. Netzer and T. Wang. Reading digits in natural images with unsupervised feature learning, 2011.

[14] A. v. d. Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. Conditional image generation with pixelcnn decoders. *arXiv preprint arXiv:1606.05328*, 2016.

[15] R. Peharz, S. Lang, A. Vergari, K. Stelzner, A. Molina, M. Trapp, G. Van den Broeck, K. Kersting, and Z. Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *International Conference on Machine Learning*, pages 7563–7574. PMLR, 2020.

[16] R. Peharz, A. Vergari, K. Stelzner, A. Molina, X. Shao, M. Trapp, K. Kersting, and Z. Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Uncertainty in Artificial Intelligence*, pages 334–344. PMLR, 2020.

[17] H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011.

[18] E. Richardson and Y. Weiss. On GANs and GMMs. *Advances in Neural Information Processing Systems*, 2018-December(NeurIPS):5847–5858, 2018.

[19] Y. Tang, R. Salakhutdinov, and G. Hinton. Deep mixtures of factor analysers. *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, 1:505–512, 2012.

[20] M. S. Tanveer, M. U. K. Khan, and C.-M. Kyung. Fine-tuning darts for image classification. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4789–4796. IEEE, 2021.

[21] A. Van Den Oord and B. Schrauwen. Factoring variations in natural images with deep Gaussian mixture models. *Advances in Neural Information Processing Systems*, 4(January):3518–3526, 2014.

[22] C. Viroli and G. J. McLachlan. Deep Gaussian mixture models. *Statistics and Computing*, 29(1):43–51, 2019.

[23] J. Wolfshaar and A. Pronobis. Deep generalized convolutional sum-product networks. In *International Conference on Probabilistic Graphical Models*, pages 533–544. PMLR, 2020.

[24] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. pages 1–6, 2017.

IX. Appendix A: DCGMM architectures

Precise layer configurations of the various DCGMM instances used in this article are given in Tab. VI. As a rule, cGMM components were chosen as high as possible while respecting memory constraints during training. All cGMM layers are assumed to use parameter sharing, otherwise the number of trainable parameters will be higher. We observe that the number of trainable parameters actually decreases as more cGMM layers are added. As with CNNs, this is because the number of parameters mainly scales with the filter size of folding layers, which can be kept small in deep architectures.

ID	Configuration	parameters
A	F(28,1)-(49)	38416
B	F(8,2)-**G**(49)-F(11,1)-**G**(49)	293657
C	F(8,1)-**G**(49)-P(2,2)-**G**(49)	243236
D	F(3,1)-**G**(25)-P(2,2)-F(4,1)-**G**(25)-P(2,2)-F(5,5)-**G**(49)	40850
E	F(3,1)-**G**(25)-F(4,2)-**G**(25)-F(12,1)-**G**(49)	186625
F	F(3,1)-**G**(25)-F(4,2)-**G**(25)-F(4,2)-**G**(25)-F(5,1)-**G**(49)	50850
G	F(3,1)-**G**(25)-P(2,2)-F(3,1)-**G**(25)-P(2,2)-F(3,1)-. . . . . G(25)-P(2,2)-F(2,1)-**G**(49)	16375

TABLE VI
Overview of DCGMM configurations used in the experiments. Layer types are F (half-convolution layer), G (cGMM layer) and P(max-pooling layer). Optionally, a linear classifier layer can be added at the top of each instance for conditional sampling.

X. Appendix B: CNN for measuring conditional sampling

The CNN classifier used in Sec. VI-F has a layer structure as given in Tab. VII. It is implemented in TensorFlow2/Keras and trained for 15 Epochs on either MNIST or FashionMNIST, using an Adam optimizer, a learning rate of 0.01 and a batch size of 100. For MNIST, this is sufficient for state-of-the-art performance ($> 99\%$), whereas for FashionMNIST, a performance of roughly 91% can be reached. While this is inferior to the performance obtained by more refined models (e.g., [20]), we accept it here for simplicity, and also because the CNN classifier is just a tool to detect differences in the distributions of real and generated samples.

Type	Kernel	Prob.	channels/neurons
Dropout		0.1	
Conv/ReLU	3		64
Conv/ReLU	3		64
Pooling	2		
Conv/ReLU	3		64
Pooling	2		
Conv/ReLU	3		64
Pooling	2		
Dense/ReLU			350
Dropout		0.1	
Dense/ReLU			350
Dense/ReLU			350
Dense/Softmax			10

TABLE VII
Hyper-Parameters of the CNN used for assessing sampling performance.

XI. Appendix C: DCGMM sharpening for FashionMNIST

Sharpening behaves similarly for the FashionMNIST dataset as it was found for MNIST in Sec. VI-D. Namely, the shallower DCGMM-B instance profits strongly from a sharpening through a single max-pooling layer, but sees deterioration of sampling performance when more max-pooling layers are involved, as in instance DCGMM-D. Fig. 9 shows this quite nicely.

Appendix D: More details on outlier detection

Outlier detection in quantified using a ROC-like curve, plotting kept inliers against rejected outliers while varying the separation threshold that is applied to the log-likelihoods. Typical examples of such curves are shown in Fig. 10.

Appendix E: Variational autoencoder details

Table Tab. VIII gives details about the convolutional VAE used to generate SVHN samples. It was trained for 100 epochs on all SVHN classes using the Adam optimizer and a learning rate of 0.0001.

Type	Kernel	Stride	channels/size
Encoder			
Conv/ReLU	3	1	64
Conv/ReLU	3	2	128
Conv/ReLU	3	2	256
Conv/ReLU	2	2	512
Flatten	-	-	4096
Dense			2*64
Decoder			
Dense/ReLU			4*4*128
Reshape			4,4,128
Conv2D^T/ReLU	4	1	1024
Conv2D^T/ReLU	5	1	512
Conv2D^T/ReLU	4	2	256
Conv2D^T/ReLU	5	1	128
Conv2D^T/ReLU	3	1	128
Conv2D^T/ReLU	3	1	3

TABLE VIII
Hyper-Parameters of the VAE used for SVHN sampling.

XII. Appendix F: Additional sampling results

. This appendix gives MNIST sampling results in a more complete fashion, that is, including more DCGMM instances, in Fig. 11. Fig. 12 gives samples from the same DCGMM instances for FashionMNIST, whereas Fig. 13 shows FashionMNIST samples generated by SPNs.

XIII. Appendix G: Visual alphabet

The lower cGMM layers of a cGMM instance usually model small image patches extracted by a preceding folding layer. With parameter sharing enabled, the cGMM therefore describes all positions within an image using a single set of parameters. This makes the most sense for low-hierarchy layers, since local image content tends to be similar across image at small patch sizes. The cGMM prototypes there form a kind of "visual alphabet", a set of centroids that, together, best describe local image content. We exemplarily show this

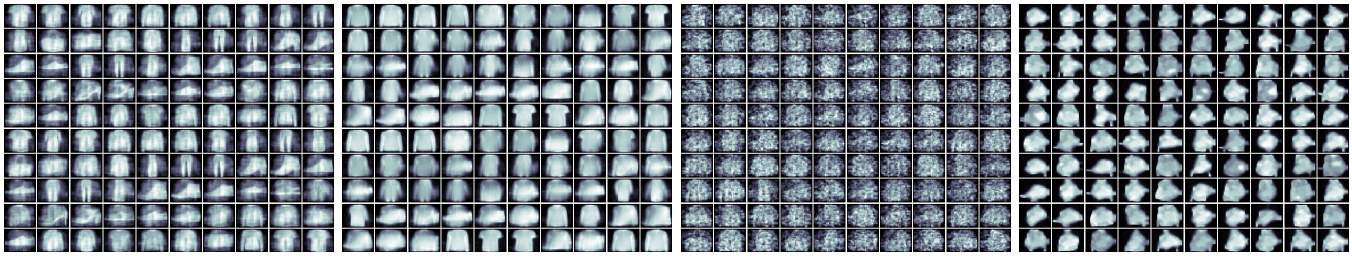

Fig. 9. Effects of sharpening for two different DCGMM instances. From left to right: DCGMM-B(no sharpening), DCGMM-B(sharpening), DCGMM-D(no sharpening), DCGMM-D (sharpening).

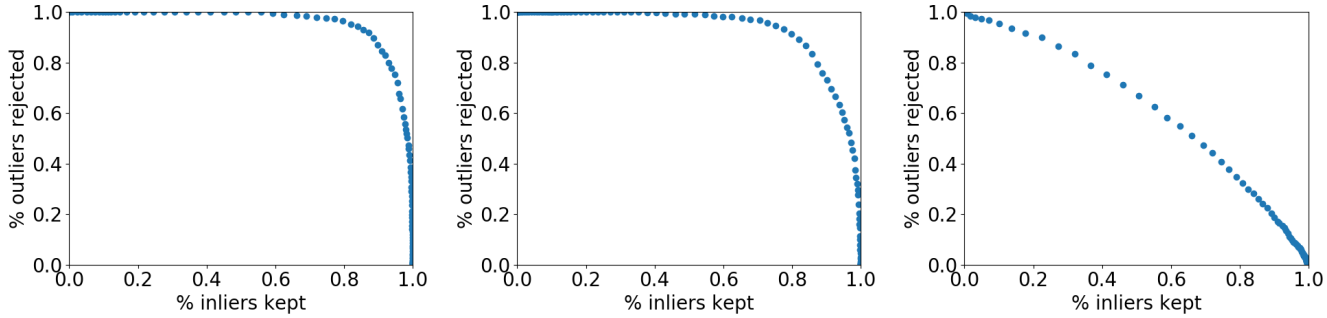

Fig. 10. ROC-like curves for outlier detection on MNIST. Left: DCGMM-A (MNIST), middle: DCGMM-E (MNIST), right: DCGMM-A(FashionMNIST). The area under these curves is taken to be a measure of outlier detection capacity.

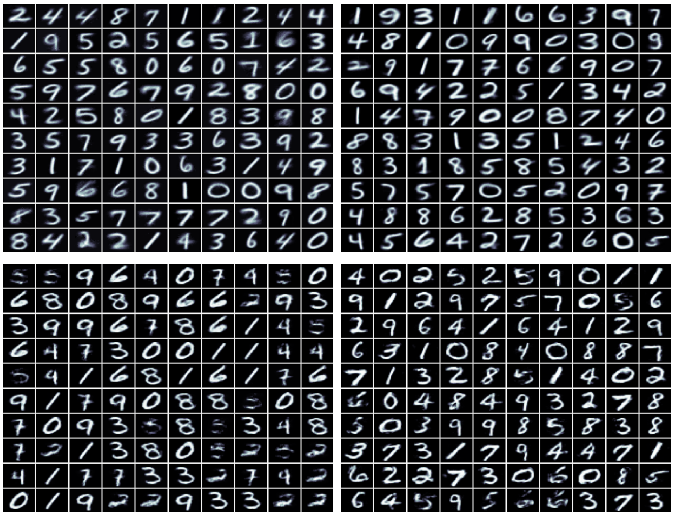

Fig. 11. Samples from several DCGMM instances for MNIST. Upper row: DCGMM-A(left), DCGMM-B(right). Lower row: DCGMM-E(left), DCGMM-F(right).

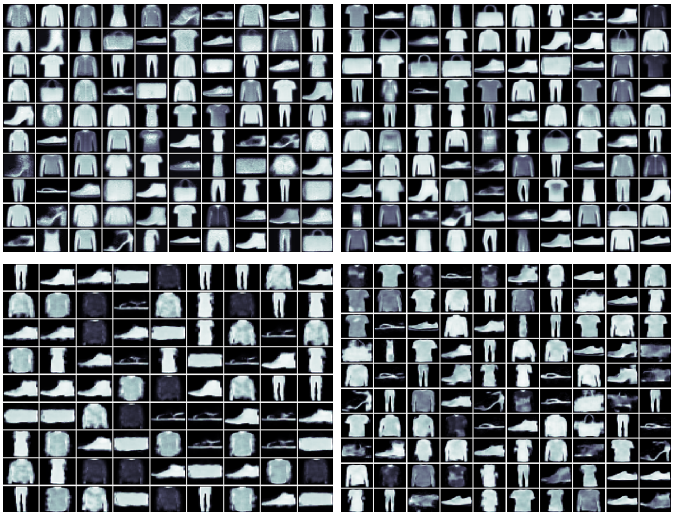

Fig. 12. Samples from several DCGMM instances for FashionMNIST. Upper row: DCGMM-A (left), DCGMM-B (right). Lower row: DCGMM-E (left), DCGMM-F (right).

for MNIST and FashionMNIST by visualizing the centroids of the lowest cGMM layer in instance DCGMM-B, which models 8x8 image patches. We observe in Fig. 14 that the basic building blocks of both datasets are faithfully represented.

XIV. Appendix H: Used libraries

For implementing RAT-SPN and PD-SPN, we made use of the public code provided under https://github.com/cambridge-mlg/EinsumNetworks which relies mainly on PyTorch. DGCSPNs are implemented using *libspn-keras* which TensorFlow2-based and can be obtained from https://github.com/pronobis/libspn-keras. VAEs and CNNs are self-implemented in TensorFlow2/Keras. TensorFlow2-Code for DCGMM can be found under https://github.com/anon-scientist/iclr22-submission.

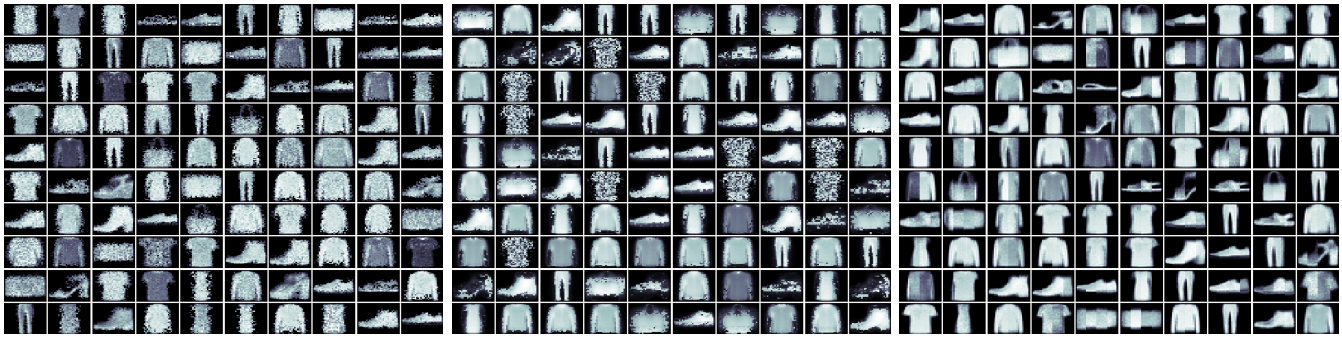

Fig. 13. SPN samples for FashionMNIST. From left to right: RAT-SPN, PD-SPN, DGCSPN.

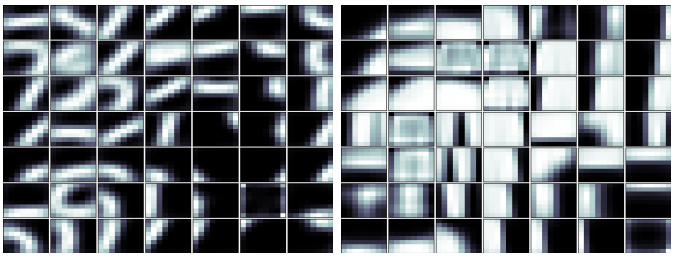

Fig. 14. Visualization of centroids in the lowest cGMM layer of DCGMM-B.
Left: training on MNIST, right: training on FashionMNIST.