# Image Modeling with Deep Convolutional Gaussian Mixture Models

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

Benedikt Pflb
*Fulda University of Applied Sciences*
Fulda, Germany
benedikt.pfuelb@cs.hs-fulda.de

***Abstract*—In this conceptual work, we present Deep Convolutional Gaussian Mixture Models (DCGMMs), a new formulation of deep hierarchical Gaussian Mixture Models (GMMs) that is particularly suited for describing and generating images. Vanilla (i.e., flat) GMMs require a very large number of components to well describe images, leading to long training times and memory issues. DCGMMs avoid this by a stacked architecture of multiple GMM layers, linked by convolution and pooling operations. This allows to exploit the compositionality of images in a similar way as deep CNNs do. DCGMMs can be trained end-to-end by Stochastic Gradient Descent. This sets them apart from vanilla GMMs which are trained by Expectation-Maximization, requiring a prior k-means initialization which is infeasible in a layered structure. For generating sharp images with DCGMMs, we introduce a new gradient-based technique for sampling through non-invertible operations like convolution and pooling. Based on the MNIST and FashionMNIST datasets, we validate the DCGMMs model by demonstrating its superiority over flat GMMs for clustering, sampling and outlier detection.**

***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, whose main objectives are both density estimation and image generation (sampling). Since images usually do not precisely follow a Gaussian mixture distribution, such a treatment is inherently approximative in nature. Image generation is currently very active research topic, and similar techniques are being investigated for generating videos [9], [25].

An issue with many recent approaches is the lack of density estimation capacity, i.e., explicitly expressing the learned probability-under-the-model $p(\boldsymbol{x})$ of an image $\boldsymbol{x}$.

In contrast, GMMs explicitly describe the distribution $p(\boldsymbol{X})$, given by a set of training data $\boldsymbol{X} = \{\boldsymbol{x}_n\}$, as a weighted mixture of $K$ Gaussian component densities $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \equiv \mathcal{N}_k(\boldsymbol{x})$:

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

Conceptually, GMMs are latent-variable models: it is assumed that the unobservable (latent) variable $z$ determines from which component a data vector $\boldsymbol{x}$ has been sampled, which is expressed as

$$p(\boldsymbol{x}, z) = \pi_z \mathcal{N}_z(\boldsymbol{x}) \tag{2}$$

Marginalizing the latent variable in Eq. (2), we obtain Eq. (1).

GMMs require the mixture weights to be normalized: $\sum_k \pi_k = 1$ and the covariance matrices to be positive definite: $\boldsymbol{x}^T \boldsymbol{\Sigma}_k \boldsymbol{x} > 0 \ \ \forall \boldsymbol{x}$. The quality of the current fit-to-data is expressed by the (incomplete) log-likelihood

$$\mathcal{L}(\boldsymbol{X}) = \mathbb{E}_n \left[ \log \sum_k \pi_k \mathcal{N}_k(\boldsymbol{x}_n) \right], \tag{3}$$

which is what GMM training optimizes, usually by variants of Expectation-Maximization (EM) [4]. It can be shown that any distributions can, given enough components, be approximated by mixtures of Gaussians [11]. Thus, GMMs are guaranteed to model the complete data distribution, but only to the extent allowed by the number of components $K$.

In this respect, GMMs are similar to flat neural networks with a single hidden layer: although, by the universal approximation theorem of [26] and [14], they *can* approximate arbitrary functions (from certain rather broad function classes), they fail to do so in practice. The reason for this is that the number of required hidden layer elements is unknown, and usually beyond the reach of any reasonable computational capacity. For images, this problem was largely solved by introducing deep Convolutional Neural Networks (CNNs). CNNs model the statistical structure of images (hierarchical organization and translation invariance) by chaining multiple convolution and pooling layers. Thus the number of parameters can be reduced without compromising accuracy .

### A. Objective, Contribution and Novelty

The objectives of this article are to introduce a GMM architecture which exploits the same principles that led to the performance explosion of CNNs. In particular, the genuinely novel characteristics are:

- conceptually new formulation of deep GMMs, including convolution and pooling operations
- scalable end-to-end training by SGD from random initial conditions (no k-means initialization)
- efficient large-scale training on high-demensional images

- realistic sampling despite non-invertible operations (pooling, convolutions)
- better empirical performance than vanilla GMMs on images for sampling, clustering and outlier detection

In addition, we provide a publicly TensorFlow implementation which supports a Keras-like flexible construction of DCGMM instances.

### B. Related Work

**GANs and VAEs** The currently most widely used models of image modeling and generation are GANs [1], [12], [22] and VAEs. Generative Adverserial Networks (GANs) are capable of sampling photo-realistic images [28], but are unable to perform density estimation since there is no way to obtain $p(\boldsymbol{x})$ for a given sample $\boldsymbol{x}$. Furthermore, their probabilistic interpretation remains unclear since they do not possess a differentiable loss function that is minimized by training. They may suffer from what is termed *mode collapse*, which is hard to detect automatically due to the absence of a loss function [28]. Variational Autoencoders (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. Similar approaches, with similar strength in sampling but lack of density estimation, are realized by the PixelCNN architecture [23] and GLOW [18] and their variants.

**Hierarchical GMMs** A straightforward hierarchical extension of GMMs is presented by [20] with the goal of unsupervised clustering: responsibilities of one GMM are treated as inputs to a subsequent GMM, together with an adaptive mechanism that determines the depth of the hierarchy. [6] present a comparable, more information-theoretic approach but not targetting sampling either. The closest related work the model we present here is described in [30]–[32]. All of these models perceive hierarchical GMMs as modular decompositions of GMMs that are really flat. The conceptual foundation is that sampling from a GMM implies the transformation of a normally distributed latent variable $\boldsymbol{z}$ by a single GMM component $k$, with weight $\pi_k$, $\boldsymbol{z}$ as $\boldsymbol{y} = \boldsymbol{A}_k\boldsymbol{z} + \boldsymbol{\beta}_k + \epsilon_k$. This leads to a distribution $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\beta}_k, \boldsymbol{A}_k\boldsymbol{A}_k^T + \epsilon_k)$. The choice which GMM component is allowed to transform the latent variable is based on component weights $\pi_k$. Hierarchical GMMs are now realized by *several* such sampling steps (layers), the sampling results of the previous layer representing the latent variable to be transformed by the next one. None of these models consider convolutional or max-pooling operations which have been proven to be important for modeling the statistical structure of images.

**Mixture of Factor Analyzers (MFAs)** MFAs models [8], [21] can be considered as hierarchical GMMs because they are formulated in terms of a lower-dimensional latent-variable representation, which is mapped to a higher-dimensional space. The use of MFAs for describing natural images is discussed in detail in [28], showing that the MFA model alone, without further hierarchical structure, compares quite favorably to GANs when considering image generation.

**Hierarchical Mixture Models** An interesting overview of the current research landscape in terms of hierarchical generative mixture models is given in [16]. 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. Principal competitors in this domain are Sum-Product Networks (SPNs, see, e.g., [27]), which are tree structures of arbitrary depth, with leaves that represent tractable distribution families (typically multi-variate normal, binomial or student-t distributions). SPN nodes perform either weighted summation or multiplication, and with appropriate constraints on the tree structure it can be shown that sampling, inference and density estimation remain tractable. Problems include finding a suitable SPN structure, and dealing with complex and high-dimensional data. Convolutional extensions of the SPN model exist, although they have been applied to very small images only [2]. A similar approach is represented by probabilistic circuits (PCs, see, e.g., [24]) or Tensorial Mixture Models (TMMs, see, e.g., [29]).

**Convolutional GMMs** The only work we could identify proposing hierarchical convolutional GMMs is [10], although the article describes a hybrid model where a CNN and a GMM are combined.

**SGD and End-To-End Training for GMMs** Training GMMs by Stochastic Gradient Descent (SGD) is challenging due to local optima and the need to enforce model constraints, most notably the constraint of positive-definite covariance matrices. At the same time, SGD is attractive in deep architectures due to its simplicity, which is facilitated even further by modern machine learning packages that perform automatic differentiation. SGD for GMMs has recently been discussed in [15], although the proposed solution requires parameter initialization by k-means and introduces several new hyper-parameters. Thus, it is unlikely to work as-is in a hierarchical structure. An SGD approach that achieves robust convergence even without k-means-based parameter initialization is presented by [7]. Undesirable local optima caused by random parameter initialization are circumvented by an adaptive annealing strategy. SPNs and PCs can be trained end-to-end by SGD as well, although no article describes this in detail. It is presumable that data-driven initialization is required here, as well. Previous hierarchical GMM proposals [30]–[32] use (quite complex) extensions of the EM algorithm initialized by k-means for training. In [28], training is performed using SGD, although with a k-means initialization.

## II. Datasets

For the evaluation we use the following image datasets:

**MNIST** [19] is the common benchmark for computer vision systems and classification problems. It consists of $60\,000$ $28 \times 28$ gray scale images of handwritten digits (0-9).

**FashionMNIST** [33] consists of images of clothes in 10 categories and is structured like the MNIST dataset.

Although these datasets are not particularly challenging for classification, their dimensionality of $784$ is at least one magni-

tude higher than datasets used for validating other hierarchical GMM approaches in the literature.

## III. DCGMM: Model Overview

The Deep Convolutional Gaussian Mixture Model (DCGMM) is a hierarchical model consisting of *layers* in analogy to CNNs.[1] Each layer with index $L$ expects an input tensor $\mathbf{A}^{(L-1)} \in \mathbb{R}^4$ of dimensions $N, H^{(L-1)}, W^{(L-1)}, C^{(L-1)}$ and produces an output tensor $\mathbf{A}^{(L)} \in \mathbb{R}^4$ of dimensions $N, H^{(L)}, W^{(L)}, C^{(L)}$. Layers can have internal variables $\boldsymbol{\theta}^{(L)}$ that are adapted during SGD training.

An DCGMM layer $L$ has two basic operating modes (see Fig. 1): for (density) *estimation*, an input tensor $\mathbf{A}^{(L-1)}$ from layer $L-1$ is transformed into an output tensor $\mathbf{A}^{(L)}$. For *sampling*, the direction is reversed: each layer receives a control signal $\mathbf{T}^{(L+1)}$ from layer $L+1$ (same dimensions as $\mathbf{A}^{(L)}$), which is transformed into a control signal $\mathbf{T}^{(L)}$ to layer $L-1$ (same dimensions as $\mathbf{A}^{(L-1)}$).

### A. Layer Types

We define four layer types: Folding (F), Pooling (P), Linear Classifier (C) and GMM (G). Each implements distinct operations for both modes, i.e., estimation and sampling.

*1) Folding Layer:* For density estimation, this layer performs a part of the well-known convolution operation known from CNNs. Based on the filter sizes $f_X^{(L)}$, $f_Y^{(L)}$ as well as the filter strides $\Delta_X^{(L)}$, $\Delta_Y^{(L)}$, all entries of the input tensor inside the range of the sliding filter window are dumped into the channel dimension of the output tensor. We thus obtain an output tensor of dimensions $N, H^{(L)} = 1 + \frac{H^{(L-1)} - f_Y^{(L)}}{\Delta_Y^{(L)}}$, $W^{(L)} = 1 + \frac{W^{(L-1)} - f_X^{(L)}}{\Delta_X^{(L)}}$ and $C^{(L)} = C^{(L)} f_X^{(L)} f_Y^{(L)}$, whose entries are computed as $\mathbf{A}^{(L)}_{nhwc} = \mathbf{A}^{(L-1)}_{nh'w'c'}$ with $h = h'/f_Y^{(L)}$, $w = w'/f_X^{(L)}$ and $c = c' + \big((h' - h\Delta_Y^{(L)}) f_X^{(L)} + w' - w\Delta_X^{(L)}\big) C^{(L-1)} + c'$. When sampling, it performs the inverse mapping which is not a one-to-one correspondence: input tensor elements which receive several contributions are averaged over all contributions.

*2) Pooling Layer:* For density estimation, pooling layers perform the same operations as standard (max-)pooling layers in CNNs based on the kernel sizes $k_Y^{(L)}$, $k_X^{(L)}$ and strides $\Delta_X^{(L)}$, $\Delta_Y^{(L)}$. When sampling, pooling layers perform a simple nearest-neighbor up-sampling by a factor indicated by the kernel sizes and strides.

*3) GMM Layer:* This layer type contains $K$ GMM components, each of which is associated with trainable parameters $\pi_k$, $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$, $k = 1 \dots K$, representing the GMM *weights*, *centroids* and *covariances*. What makes GMM layers convolutional is that they do not model single input vectors, but the channel content at *all* positions $h, w$ of the input $\mathbf{A}^{(L-1)}_{n,w,h,:}$, using a shared set of parameters. This is analog to the way a CNN layer models image content at all sliding window positions using the same filters. A GMM layer thus maps the input tensor $\mathbf{A}^{(L-1)} \in \mathbb{R}^{N, H^{(L-1)}, W^{(L-1)}, C^{(L-1)}}$ to $\mathbf{A}^{(L)} \in \mathbb{R}^{N, H^{(L-1)}, W^{(L-1)}, K}$, each GMM component $k \in \{1, \dots, K\}$ contributing the likelihood $\boldsymbol{A}^{(L)}_{nhwk}$ of having generated the channel content at position $h, w$ (for sample $n$ in the mini-batch). This likelihood is often referred to as *responsibility* and is computed as

$$\begin{aligned} p_{nhwk}\big(\mathbf{A}^{(L-1)}\big) &= \mathcal{N}_k\big(\mathbf{A}^{(L-1)}_{nhw:}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\big) \\ \boldsymbol{A}^{(L)}_{nhwk} &\equiv \frac{p_{nhwk}}{\sum_{c'} p_{nhwc'}}. \end{aligned} \tag{4}$$

For training the GMM layer, we optimize the GMM log-likelihood $\mathcal{L}^{(L)}$ for each layer $L$:

$$\begin{aligned} \mathcal{L}^{(L)}_{hw} &= \sum_n \log \sum_k \pi_k p_{nhwk}(\mathbf{A}^{(L-1)}) \\ \mathcal{L}^{(L)} &= \frac{\sum_{hw} \mathcal{L}^{(L)}_{hw}}{H^{(L-1)} W^{(L-1)}} \end{aligned} \tag{5}$$

Training is performed by SGD according to the technique, and with the recommended parameters, presented by [7], which uses a max-component approximation to $\mathcal{L}^{(L)}$. In sampling mode, a control signal $\mathbf{T}^{(L)}$ is produced by standard GMM sampling, performed separately for all positions $h, w$. GMM sampling at position $h, w$ first selects a component by drawing from a multinomial distribution. If the GMM layer is the last layer of a DCGMM instance, the multinomial's parameters are the mixing weights $\pi_:$ for each position $h, w$. Otherwise, the control signal $\mathbf{T}^{(L+1)}_{nhw:}$ received from layer $L+1$ is used. It is consistent to use the control signal for component selection in layer $L$, since it was sampled by layer $L+1$, which was in turn trained on the component responsibilities of layer $L$, see Sec. VI. The selected component (still at position $h, w$) then samples $\mathbf{T}^{(L)}_{nhw:}$. It is often beneficial for sampling to restrict component selection to the $S$ components with the highest control signal (top-$S$ sampling). This reduces the diversity of sampling (since less components can participate), but improves its quality (since selection is restricted to the most likely components).

*4) Linear Classifier Layer:* This layer type implements a linear classifier, trained in a supervised fashion by cross-entropy loss. Logits are obtained via an affine transformation $\mathbf{A}^{(L)} = \hat{\mathbf{A}}^{(L-1)} \boldsymbol{W}^{(L)} + \boldsymbol{b}^{(L)}$ from the flattened input activities $\hat{\mathbf{A}}^{(L-1)} \in \mathbb{R}^{N \times HWC}$. The logits have the dimension $N \times K$, with $K$ representing the number of classes or categories. For sampling, a control signal is generated by approximately inversing this transformation: $\mathbf{T}^{(L-1)} = \boldsymbol{W}^{(L),T} \mathbf{T}^{(L)} - \boldsymbol{b}^{(L)}$, where $\mathbf{T}^{(L)} \in \mathbb{R}^{N \times K}$ contains the one-hot coded classes for each sample to be generated.

### B. Architecture-Level Functionalities

*1) End-to-End Training:* To train an DCGMM instance, we optimize $\mathcal{L}^{(L)}$ for each GMM layer $L$ by vanilla SGD[2], using learning rates $\epsilon^{(L)}$. This is different from a standard CNN classifier, where only a single loss function is minimized,

[1] Repository: https://gitlab.cs.hs-fulda.de/ML-Projects/dcgmm

[2] Advanced SGD strategies like RMSProp [13] or Adam [17] seem incompatible with GMM optimization.

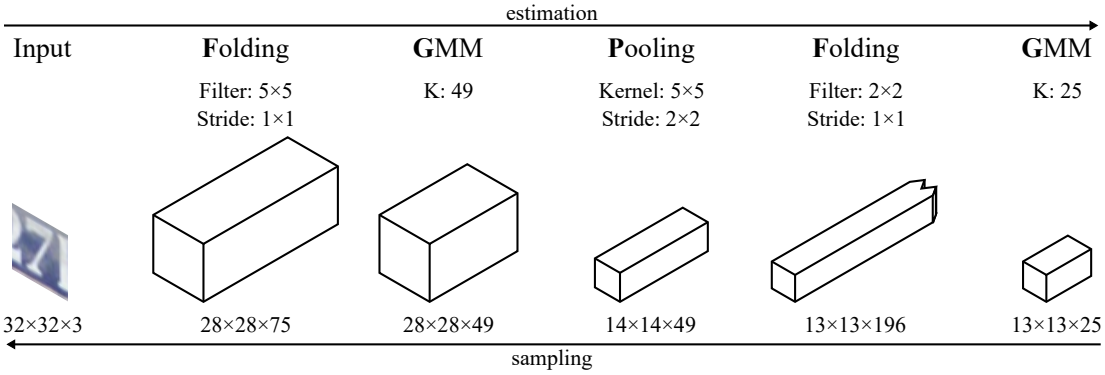


Fig. 1. Illustration of a DCGMM instance containing all layer types, with exemplary dimensionalities and parameters for each layer.

usually a cross-entropy loss computed from the last layer's outputs. Learning is *not* conducted layer-wise but end-to-end. Parameter initialization for GMM layers selects the initial values for the mixing weights as $\pi_k = K^{-1}$, centroid elements sampled from $\boldsymbol{\mu}_{kl} \sim \mathcal{U}_{[-0.01,0.01]}$ and diagonal covariances are initialized to unit entries. To ensure convergence, training is conducted in two phases. In the first phase, only centroids are adapted, whereas both centroids and precisions are adapted in the second.

*2) Density Estimation and Outlier Detection:* Outlier detection requires the computation of long-term averages in all layers and positions, $\mathbb{E}_n \mathcal{L}^{(L)}_{nhw}$ and variances $\mathrm{Var}_n(\mathcal{L}^{(L)}_{nhw})$ over the training set, preferably during a later, stable part of training. Thus, for every layer and position $h, w$, inliers are characterized by

$$\mathcal{L}^{(L)}_{hw} \geq B^{(L)}_{hw} \equiv \mathbb{E}_n \mathcal{L}^{(L)}_{nhw} - c\sqrt{\mathrm{Var}_n(\mathcal{L}^{(L)}_{nhw})}\,. \tag{6}$$

A larger $c$ implies a less restrictive identification of inliers.

Assuming that the topmost GMM layer is global ($h = w = 1$), Eq. (6) reduces to a single condition that determines whether the sample, as a whole, is an inlier. However, we can also localize inlier/outlier image parts by evaluating Eq. (6) in lower GMM layers.

*3) Sampling and Sharpening:* Sampling starts in the highest layer $L$, assumed to be an GMM layer, and propagates control signals downwards (see Fig. 1 and Sec. III-A3), with control signal $\mathbf{T}^{(L)}$ constituting the sampling result. Sampling suffers from information loss due to the non-invertible mappings effected by Pooling and Folding layers. To counteract this, a Folding or Pooling layer at $L-1$ performs *sharpening* on the control signal $\mathbf{T}^{(L-1)}$ it generates from $\mathbf{T}^{(L)}$. This involves computing $\mathcal{L}^{(L)}(\mathbf{T}^{(L-1)})$ for the GMM layer at level $L$ and performing $G$ gradient ascent steps $\mathbf{T}^{(L-1)}_{nhwc} \to \mathbf{T}^{(L-1)}_{nhwc} + \epsilon_s \partial \mathcal{L}^{(L)} / \partial \mathbf{T}^{(L-1)}_{nhwc}$. The reason for sharpening is that filters in Folding layers usually overlap, and neighboring filter results are correlated. This correlation is captured by all higher GMM layers, and most prominently by the next-highest one. Therefore, modifying $\mathbf{T}^{(L-1)}$ by gradient ascent will recover some of the information lost by pooling or folding. After sharpening, the tensor $\mathbf{T}^{(L-1)}$ is passed as control signal signal to $L-2$.

*4) Conditional Sampling:* By conditional sampling we understand the ability so selectively produce samples from a certain class. Obviously, this is possible only when the DCGMM instance has a top-level linear classifier layer. We proceed as in Sec. III-B3, except for the component selection in the top-level GMM layer. For conditional sampling, we simply train a linear classifier in the last layer $L$ on the outputs $\mathbf{A}^{(L-1)}$ of the previous layer, and then approximately invert this mapping given a certain class label to obtain a control signal $\mathbf{T}^{(L)}$ for sampling in layer $L-1$.

*5) In-Painting:* This is a functionality where a corrupted image, from which a part has been deleted, is supplied to a trained DCGMM instance, which then *completes* the missing parts. Here, we add an additional tweist by not informing the model which parts of the image have been deleted. Rather, a DCGMM should infer this on its own using its outlier detection capability, see **??**.

Mathematically, in-painting requires that we perform *posterior inference* from a trained DCGMM. We shall consider this kind of inference for a flat GMM first, and subsequently generalize to DCGMMs.

Assuming that an image $\boldsymbol{x}$ is composed of two parts $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, where $\boldsymbol{x}_2$ is corrupted. To recover it, we wish to draw samples from the distribution $p(\boldsymbol{x}_2|\boldsymbol{x}_1)$. A simple computation yield an expression that can be evaluated by re-introducing the latent variable $z$ of the GMM, see Eq. (2)):

$$p(\boldsymbol{x}_2|\boldsymbol{x}_1) = \frac{p(\boldsymbol{x}_1\boldsymbol{x}_2)}{p(\boldsymbol{x}_1)} = \sum_z \frac{p(\boldsymbol{x}_1, z)}{p(\boldsymbol{x}_1, z)} \frac{p(\boldsymbol{x}_2\boldsymbol{x}_1, z)}{p(\boldsymbol{x}_1)} = \tag{7}$$

$$= \sum_z p(\boldsymbol{x}_2|\boldsymbol{x}_1, z) p(z|\boldsymbol{x}_1) \equiv \sum_z \gamma_z p(\boldsymbol{x}_2|\boldsymbol{x}_1, z). \tag{8}$$

This shows that the distribution we wish to sample from is again a Gaussian mixture. The mixture coefficients are given by the *responsibilities* $\gamma_z = p(z|\boldsymbol{x}_1)$, whereas the component densities can be simplified as $p(\boldsymbol{x}_2|\boldsymbol{x}_1, z) \sim p(\boldsymbol{x}_2\boldsymbol{x}_1, z)$. We first draw a value $z^*$ from a multi-nomial distribution defined by the $\gamma_i$, and then generate a sample $\boldsymbol{x}$ from component

TABLE I
CONFIGURATIONS OF DIFFERENT DCGMM ARCHITECTURES.

| ID / layer | $1L$ | $2L$-$a$ | $2L$-$b$ | $2L$-$c$ | $2L$-$d$ | $2L$-$e$ | $3L$-$a$ | $3L$-$b$ |
|---|---|---|---|---|---|---|---|---|
| 1 | F(28,28,1,1) | F(20,20,8,8) | F(7,7,7,7) | F(8,8,2,2) | F(28,28,1,1) | F(4,4,2,2) | F(3,3,1,1) | F(28,28,1,1) |
| 2 | G(25) | G(25) | G(25) | G(25) | G(25) | G(25) | G(25) | G(25) |
| 3 | | F(2,2,1,1) | F(4,4,1,1) | F(11,11,1,1) | F(1,1,1,1) | F(13,13,1,1) | P(2,2) | F(1,1,1,1) |
| 4 | | G(36) | G(36) | G(36) | G(36) | G(36) | F(4,4,1,1) | G(25) |
| 5 | | | | | | | G(25) | F(1,1,1,1) |
| 6 | | | | | | | P(2,2) | G(25) |
| 7 | | | | | | | F(6,6,1,1) | |
| 8 | | | | | | | G(49) | |
| *comment* | vanilla GMM | 1 conv. layer | 1 conv. layer | 1 conv. layer | no convolutions | 1 conv. layer | 2 conv. layers | no convolutions |

probability $p(\boldsymbol{x}_2\boldsymbol{x}_1, z^*)$ of the Gaussian mixture. The part of $\boldsymbol{x}$ that belongs to $\boldsymbol{x}_1$ is simply clamped when sampling.

In a DCGMM, we implement this procedure for every position $(h, w)$ in all GMM layers. We start at the top of the hierarchy (layer $O$), where responsibilities are simply the activities $\mathbf{A}^{(O)}$ computed by forward-propagating the corrupted input in estimation mode. The responsibilities are used to draw $z^*(h, w)$ for each position $(h, w)$ as detailed above. Using $z^*$, we obtain a control signal for layer $O-1$ as

$$\mathbf{T}^{(O)}_{hw:} \sim \mathcal{N}_{z^*(h,w)}(\cdot), \tag{9}$$

which propagated downwards through pooling and folding layers to the next-deeper GMM layer.

Since, for each GMM layer $L < O$, we want to in-paint only those parts which are not corrupted, we combine the raw control signal $\mathbf{T}^{(L)}$ and the activities $\mathbf{A}^{(L)}$ obtained in the forward pass into a fused control signal

$$\hat{\mathbf{T}}^{(L)}_{nhw:} = \begin{cases} \mathbf{A}^{(L)}_{nhw:} & \text{if inlier at } h, w \\ \mathbf{T}^{(L)}_{nhw:} & \text{if outlier at } h, w \end{cases} \tag{10}$$

where outliers are determined as detailed in Sec. III-B2. Thus, control signals fill in the activities that seem corrupted.

*6) Variant generation:* Variant generation is a special case of sampling, where a template image is supplied and the DCGMM generates similar ones. This requires forward-propagating the template image in estimation mode through the DCGMM, and subsequently sampling a control signal from the top-level GMM layer $O$ according to the obtained top-level activities $\mathbf{A}^{(O)}_{hw:}$. For all layers $L < O$, sampling is performed as detailed in Sec. III-B3. An interesting option is to control the similarity to the template image: this can be done either by selecting a suitable $S$ for top-$S$-sampling, or by clamping top-down signals to activities in higher DCGMM layers:

$$\mathbf{T}^{(L)} \equiv \mathbf{A}^{(L-1)} \text{ for } L \geq \hat{L}, \tag{11}$$

where $\hat{L} >=$ is a free parameter. The closest match between template image and generated image is to be expected for $\hat{L} = 0$, whereas $\hat{L} > O$ should result in very diverse results.

## IV. EXPERIMENTS

We define various DCGMM instances (with 2 or 3 GMM layers) for evaluation, see Tab. I, plus a single-layer DCGMM baseline which is nothing but a vanilla GMM. A DCGMM instance is defined by the parameters of its layers: **F**olding($f_Y$, $f_X$, $\Delta_Y$, $\Delta_X$), (Max-)**P**ooling($k_Y$, $k_X$, $\Delta_Y$, $\Delta_X$) and **G**MM($K$). Unless stated otherwise, training is always conducted for 25 epochs, using the recommended parameters from [7]. Sharpening is always performed for $G = 1\,000$ iterations with a step size of $0.1$.

### A. Sampling, Sparsity and Interpretability

We show that trained DCGMM parameters are sparse and have a intuitive interpretation in terms of sampling. To this effect, we train DCGMM instance $2L$-$a$ (see Tab. I). After training (see Sec. III-B1), we plot and interpret the centroids of the GMM layers 2 (G2) and 4 (G4). The centroids of layer 2 (left of Fig. 2) are easily interpretable and reflect the patterns that can occur in any of the $2 \times 2$ input patches to layer G2 of size $20 \times 20$. The $36 = 6 \times 6$ centroids of G4 (right of Fig. 2) express typical responsibility patterns computed from each of the $2 \times 2$ input patches to G2, and can be observed to be very sparsely populated. Another interpretation of G4 centroids can be found in terms of sampling (see Sec. III-B3), which would first select a random G4 component to generate a sample of dimensions $H, W, C = 1, 1, 2 \times 2 \times 5 \times 5$ from it, and pass it on as a control signal to G2. Traversing Folding layer 3 only reshapes the control signal to dimension $H, W, C = 2, 2, 5 \times 5$, depicted in the middle of Fig. 2. This signal controls component selection in each of the $2 \times 2$ positions in G2: due to their sparsity, we can directly read off the components likely to be selected for sampling at each position. G2 thus generates a control signal whose $2 \times 2$ positions of dimensions $H, W, C = 20, 20, 1$ overlap in the input plane (this is resolved by sharpening in Folding layer 1). In this case, it is easy to see that sampling produces a particular representation of the digit zero.

### B. Outlier Detection

For outlier detection, we compare DCGMM architectures from Tab. I, using the log-likelihood of the highest layer as a criterion as detailed in Sec. III-B2. We first train a

DCGMM instance on classes 0-4, and subsequently use the trained classes for inlier- and class 5-9 for outlier-detection. We vary $c$ in the range $[-2, 2]$, resulting in different outlier and inlier percentages. Fig. 3 shows the ROC-like curves thus clearly indicate that the deep convolutional DCGMM instances perform best, whereas deep but non-convolutional instances like $2L$-$d$ and $3L$-$b$ consistently perform badly.

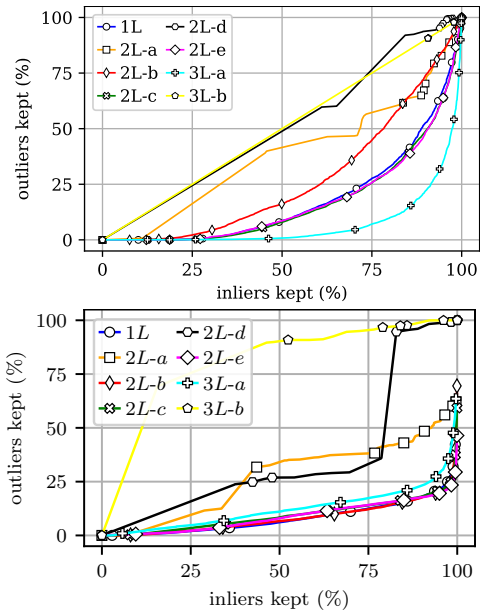


Fig. 3. Visualization of outlier detection capabilities (different DCGMMs) for MNIST (left) and FashionMNIST (right).

## C. Clustering

We compare DCGMMs to vanilla GMMs using established clustering metrics, namely the Dunn index [5] and the Davies-Bouldin score [3]. The DCGMM instances from Tab. I are tested on both image datasets. We observe (see Tab. II and Fig. 3) that mainly the deep but non-convolutional DCGMM instances perform well in clustering, whereas convolutional instances, even if they are deep, are compromised. These metrics do not measure the classification accuracy obtained by clustering but intrinsic clustering-related properties.

TABLE II
TWO METRICS, DUNN INDEX (HIGHER IS BETTER) AND DAVIES-BOULDIN (DB) SCORE (SMALLER IS BETTER), EVALUATED FOR ALL TESTED DCGMM ARCHITECTURES ON MNIST AND FASHIONMNIST. BEST RESULTS ARE MARKED IN BOLD. THE GIVEN NUMBERS ARE WORST CASES OVER 10 RERUNS.

| **Dataset** | Metric \ DCGMM | $1L$ | $2L$-$a$ | $2L$-$b$ | $2L$-$c$ | $2L$-$d$ | $2L$-$e$ | $3L$-$a$ | $3L$-$b$ |
|---|---|---|---|---|---|---|---|---|---|
| MNIST | Dunn index | 0.14 | 0.14 | 0.13 | 0.12 | **0.19** | | 0.15 | 0.15 |
| | DB score | 2.59 | 2.73 | 3.06 | 2.62 | 2.57 | | 2.65 | **2.53** |
| Fashion-MNIST | Dunn index | 0.14 | 0.15 | **0.16** | 0.15 | 0.11 | 0.11 | 0.096 | 0.13 |
| | DB score | 2.37 | 2.77 | 2.62 | 2.7 | 2.40 | 2.92 | 3.2 | **2.35** |

## D. Sampling and Sharpening

The results presented here were obtained by training on classes 0-4 of both datasets, and have to be confirmed by visual inspection of generated samples. The restriction to classes 0-4 is purely for visualization purposes.

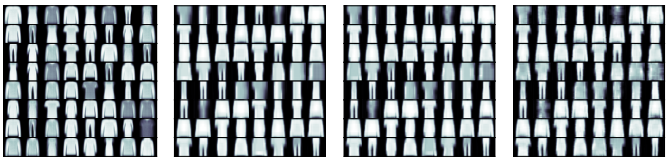

Fig. 5. Convolutional architecture is helpful for sampling: top-1 sampling shown on FashionMNIST, for DCGMM architectures $1L$ (vanilla GMM, upper left), $2L$-$d$ (non-convolutional 2-layer DCGMM, upper right), $2L$-$c$ and $2L$-$e$ (both convolutional 2-layer DCGMMs). Please observe duplicated samples in the non-convolutional architectures.

*a) Effects of convolutional architectures:* An important property of DCGMM is the ability to analyze *local* input patches. In this experiment, we evaluate sampling for various two-layer architectures with different local patch (i.e., convolution filter) sizes: global ($2L$-$d$), large ($2L$-$a$), semi-local ($2L$-$c$) and local ($2L$-$e$) and observe effects when performing top-1-sampling. The results of Fig. 4 indicate that, as filter sizes decrease, the diversity but also the sharpness of generated samples increase, at essentially no additional computational cost. Analogous FashionMNIST results are given in Fig. 5.

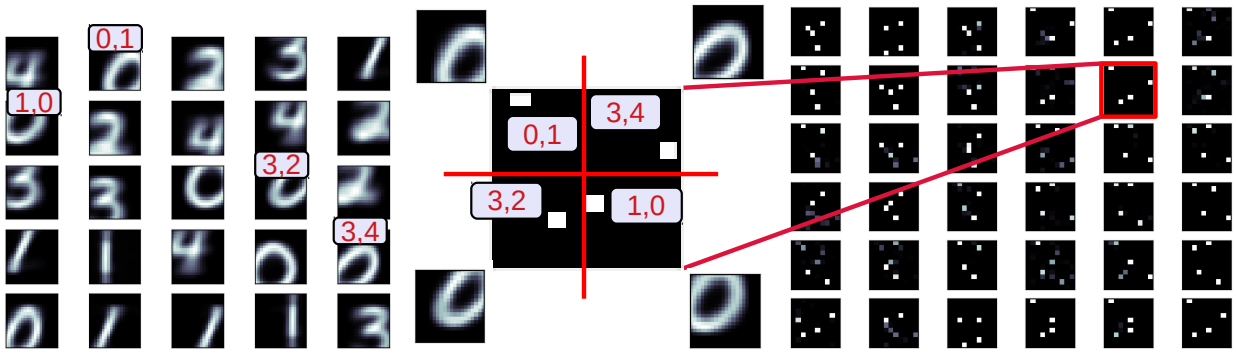


Fig. 2. Sampling from DCGMM instance $2L$-$a$, see Tab. I. Shown are learned GMM centroids (left: G2, right: G4) and an illustration of sampling, having initially selected the layer 4 component highlighted in red. In the middle, the selected G4 centroid is shown.

Fig. 4. Convolutional architecture is helpful for sampling: top-1 sampling shown on MNIST for DCGMM architectures $1L$ (vanilla GMM, upper left), $2L$-$d$ (non-convolutional 2-layer, upper right), $2L$-$c$ (convolutional 2-layer, lower left) and $2L$-$e$ (convolutional 2-layer, lower right). Please observe duplicated samples in the non-convolutional architectures, marked by identical red letters. This figure is larger than Fig. 5 so duplicated samples can be better observed.

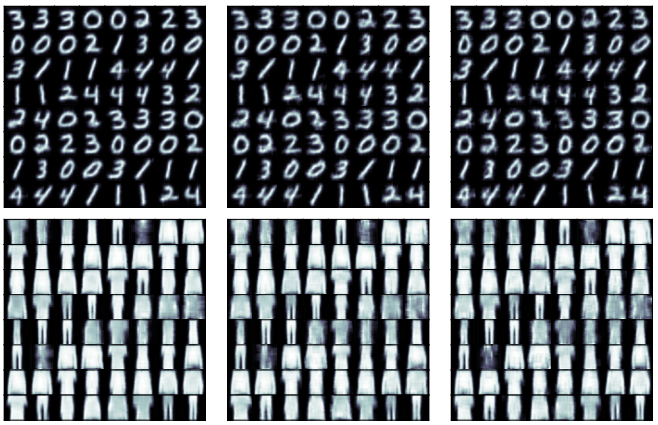

Fig. 6. Impact of higher values of $S$ in top-S sampling, shown for DCGMM instance $2L$-$c$ for MNIST and FashionMNIST. From left to right: $S$=2,5,10.

*b) Controlling Diversity by Top-S-Sampling:* Using instance $2L$-$c$, Fig. 7 demonstrates how sample diversity is related to $S$: a higher value yields more diverse samples, but increases the risk of generating corrupted samples or outliers. As the FashionMNIST results show, a good value of $S$ is clearly problem-dependent.

*c) Generating Sharp Images:* Fig. 7 shows the effect of sharpening for DCGMM instance $2L$-$c$ using top-1-sampling. We can observe that the overall shape of a sample is not changed but that the outlines are crisper, an effect visible

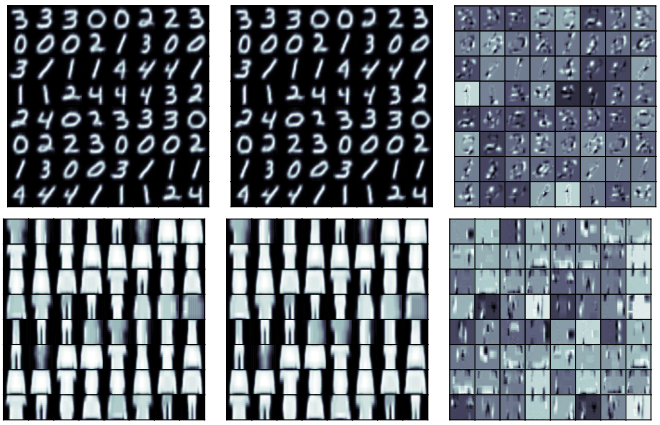

Fig. 7. Impact of sharpening on top-S-sampling with $S$=1, see Sec. III-B3, shown for DCGMM instance $2L$-$c$. Shown are unsharpened samples (left), sharpened samples (middle) and differences (right). Samples at the same position were generated by the same top-level prototype.

especially for FashionMNIST. Thus, sharpening does no harm and rather improves the visual quality of generated samples.

## V. Probabilistic Interpretation of DCGMMs

A probabilistic interpretation of the DCGMM model is possible despite its complex structure. The simple reason is that DCGMM instances produce outputs which are inherently normalizable, meaning that the integral over an infinite domain (e.g., data space) remains finite. Thus, DCGMM outputs can be interpreted as a probability which is not the case for DNN/CNNs due to the use of scalar products.

Here, we prove that GMMs are normalizable in the sense that the integral of the log-probability $\mathcal{L}(\boldsymbol{x}) = \log \sum_k \pi_k p_k(\boldsymbol{x})$ is finite. This holds for any GMM layer in a hierarchy regardless of its input, provided that the input is finite (which is assured because Pooling and Folding layers cannot introduce infinities). For simplicity, we integrate over the whole $d$-dimensional space $\mathbb{R}^d$. Since the component probabilities are Gaussian and thus strictly positive, and since furthermore the mixing weights are normalized and $\geq 0$, the sum is strictly positive. Thus, it is sufficient to show that the integral over the inner sum (the argument of the logarithm) is finite. We thus have

$$\begin{aligned} \int_{\mathbb{R}^d} \sum_k \pi_k p_k(\boldsymbol{x}) d\boldsymbol{x} &= \sum_k \pi_k \int_{\mathbb{R}^d} p_k(\boldsymbol{x}) \\ &= \sum_k \pi_k \sqrt{\det(2\pi\Sigma)} \end{aligned} \tag{12}$$

which is trivially finite because Gaussians are normalized.

## VI. Conceptual and Practical Differences to Other Hierarchical GMM Models

In this section, we wish to outline the main conceptual differences to what we consider the closest related work, namely the hierarchical GMM models presented in [30]–[32]. Although there are some differences between these works, they share, as outlined in the introduction, the notion that a GMM

performs sampling by transforming what is termed a random *latent variable*, which usually follows a simple distribution like $\mathcal{N}(0, \boldsymbol{I})$.

To clarify a purely semantic point: In this work, and close to the original derivation of the EM algorithm for GMMs [4], we consider the latent variable $\boldsymbol{z}$ of a GMM to be the one that enters into the complete-data log-likelihood of the corresponding latent-variable model. Another point where we differ in our terminology is that, for us, lower layers are closer to the input in density estimation mode. The lowest layer has index 1, counting upwards.

On the conceptual side, the cited models all share the notion that the proposed deep GMMs have a one-to-one correspondence to a flat model, which is actually the one whose log-likelihood is optimized. Consequently, layers are not optimized independently of each other, but are intrinsically linked. This is most easily visible from the fact that the component weights in each layer are not normalized: only the aggregated weights of all sampling paths through the deep model have this property.

In contrast, our model does not assume (or require) a one-to-one correspondence to a flat GMM. The layers in our model are GMMs in their own right, having, e.g., their own, normalized weights and losses, and compute the posterior probabilities $\gamma_{nj} = p(z_j = 1|\boldsymbol{x}_n)$ of their *own* latent variables. The dependency between layers is realized simply by the fact that posterior probabilities of one GMM layer are inputs to the subsequent GMM layer (potentially after being transformed by convolution and pooling layers).

The first practical consequence of this independent layers type of formulation is that layers can be optimized independently of each other. In particular, it is not necessary to enumerate all possible paths through the deep GMM as in [31] for training, which can only be done in approximation anyway. Rather, normal GMM optimization is performed for each layer, given the outputs of previous layers. This leads to a huge gain in scalability, allowing to train deep GMMs with many layers on high-dimensional data like images in a matter of minutes.

Another consequence concerns the introduction of pooling and convolution operations: since the layers of our model are not constrained by joint a flat GMM assumption, we are free to apply arbitrary transformations to their outputs before passing these on to subsequent GMM layers. The probabilistic interpretation of our model comes from the fact that each GMM in the model is independent and thus has the probabilistic interpretation that all GMMs share.

A last consequence concerns sampling: in our formulation of deep GMMs, a single GMM layer is trained on (estimated) latent variables from a preceding layer. When sampling, it can generate a valid instance $\boldsymbol{z}$ of these latent variables. Therefore, it is reasonable to use the generated $\boldsymbol{z}$ for *selecting* the GMM component to sample from in layer $X$, instead of the $\boldsymbol{\pi}^{(X)}$. Of course, a suitable merging of both quantities is possible. Sampling as described here is thus a non-linear instead of an affine transformation, as it is modeled in related work. Nevertheless, we retain the notion of paths through the DCGMM instance, given by the components selected for sampling in each layer. Since the number of paths grows exponentially with the number of layers, deeper DCGMM instances can describe much richer sample distributions.

## VII. Summary, Discussion and Conclusion

The **Objective of the article** was to establish the conceptual foundations of deep GMM hierarchies (see also Sec. V) that leverage important mechanisms from the CNN domain. Conceptual differences and shared properties w.r.t. principal related work are discussed in VI. Convolution and pooling layers make it possible to apply the model to high-dimensional image data with off-the-shelf hardware (typical training runs take approximately 2 minutes on a GeForce GTX 1080).

**Results** show the illustration of important functionalities such as outlier detection, clustering and sampling, which no other work on hierarchical GMMs can present related to such high-dimensional image datasets. We also propose a method to generated sharp images with GMMs, which has been a problem in the past [28]. An interesting facet of our experimental results is that non-convolutional DCGMMs seem to perform better at clustering, whereas convolutional ones are better at outlier detection and sampling.

A **Key point** is the compositionality in natural images. This property is at the root of DCGMM's ability to produce realistic samples with relatively few parameters. When considering top-S-sampling in a layer $L$ with $H^{(L)}W^{(L)} = P^{(L)}$ positions, the number of distinct control signals generated layer $L$ is $S^{P^{(L)}}$. A DCGMM instance with multiple GMM layers $\{L_i\}$ can thus sample $\prod_L S^{P^{(L)}}$ different patterns, which grows with the depth of the hierarchy *and* the number of distinct positions in a layer, making a strong argument in favor of deep convolutional hierarchies such as DCGMM. This is an argument similar to the one about different paths through a hierarchical MFA model in [31], [32], although the number of paths grow more strongly for DCGMMs because sampling is performed independently for each GMM position.

**Practical advantages** over other hierarchical models such as [30]–[32] are most notably the introduction of convolution and pooling layers. Our experimental validation can therefore be performed on high-dimensional data, such as images, with moderate computational cost, instead of low-dimensional problems such as the artificial Smiley task or the *Ecoli* and related problems. Our experimental validation does not exclusively focus on clustering performance (problematic with images) but on demonstrating the capacity for realistic sampling and outlier detection. Lastly, training DCGMMs by SGD facilitates efficient parallelizable implementations, as demonstrated by the TF2 implementation we provide.

**Next steps** consist of exploring more DCGMM architectures, mainly sampling, for generating natural images.

## References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. 2017.

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

[3] David L. Davies and Donald W. Bouldin. A Cluster Separation Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, 1979.

[4] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data Via the EM Algorithm . *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[5] J. C. Dunn. A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3):32–57, 1973.

[6] Vincent Garcia, Frank Nielsen, and Richard Nock. Hierarchical Gaussian mixture model. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4070–4073. IEEE, 2010.

[7] Alexander Gepperth and Benedikt Pflb. Gradient-based training of gaussian mixture models in high-dimensional spaces, 2020.

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

[9] Farhad Ghazvinian Zanjani, Svitlana Zinger, and Peter H. N. de With. Deep convolutional gaussian mixture model for stain-color normalization of histopathological images. In Alejandro F. Frangi, Julia A. Schnabel, Christos Davatzikos, Carlos Alberola-López, and Gabor Fichtinger, editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*, pages 274–282, Cham, 2018. Springer International Publishing.

[10] Farhad Ghazvinian Zanjani, Svitlana Zinger, and Peter H. N. de With. Deep Convolutional Gaussian Mixture Model for Stain-Color Normalization of Histopathological Images. In *Miccai*, volume 1, pages 274–282. Springer International Publishing, 2018.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[12] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 3(January):2672–2680, 2014.

[13] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Lecture 6.5 - RmsProp: Divide the gradient by a running average of its recent magnitude, 2012.

[14] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[15] Reshad Hosseini and Suvrit Sra. An alternative to EM for Gaussian mixture models: batch and stochastic Riemannian optimization. *Mathematical Programming*, 181(1):187–223, 2020.

[16] Priyank Jaini, Pascal Poupart, and Yaoliang Yu. Deep homogeneous mixture models: Representation, separation, and approximation. In *NeurIPS*, pages 7136–7145, 2018.

[17] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015.

[18] Diederik P Kingma and Prafulla 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.

[19] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998.

[20] Ming Liu, Eric Chang, and Bei Qian Dai. Hierarchcial Gaussian mixture model for speaker verification. *7th International Conference on Spoken Language Processing, ICSLP 2002*, pages 1353–1356, 2002.

[21] Geoffrey McLachlan and David Peel. Mixtures of Factor Analyzers. pages 238–256, jan 2005.

[22] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. pages 1–7, 2014.

[23] Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. *arXiv preprint arXiv:1606.05328*, 2016.

[24] Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *International Conference on Machine Learning*, pages 7563–7574. PMLR, 2020.

[25] Aj Piergiovanni and Michael Ryoo. Temporal Gaussian mixture layer for videos. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5152–5161, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[26] Allan Pinkus. Approximation theory of the MLP model in neural networks. *Acta Numerica*, 8:143–195, 1999.

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

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

[29] Or Sharir, Ronen Tamari, Nadav Cohen, and Amnon Shashua. Tensorial mixture models. *arXiv preprint arXiv:1610.04167*, 2016.

[30] Yichuan Tang, Ruslan Salakhutdinov, and Geoffrey Hinton. Deep mixtures of factor analysers. *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, 1:505–512, 2012.

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

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

[33] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. pages 1–6, 2017.