

# Training Faster by Separating Modes of Variation in Batch-normalized Models

Mahdi M. Kalayeh, *Member, IEEE*, and Mubarak Shah, *Fellow, IEEE*

**Abstract**—Batch Normalization (BN) is essential to effectively train state-of-the-art deep Convolutional Neural Networks (CNN). It normalizes the layer outputs during training using the statistics of each mini-batch. BN accelerates training procedure by allowing to safely utilize large learning rates and alleviates the need for careful initialization of the parameters. In this work, we study BN from the viewpoint of Fisher kernels that arise from generative probability models. We show that assuming samples within a mini-batch are from the same probability density function, then BN is identical to the Fisher vector of a Gaussian distribution. That means batch normalizing transform can be explained in terms of kernels that naturally emerge from the probability density function that models the generative process of the underlying data distribution. Consequently, it promises higher discrimination power for the batch-normalized mini-batch. However, given the rectifying non-linearities employed in CNN architectures, distribution of the layer outputs show an asymmetric characteristic. Therefore, in order for BN to fully benefit from the aforementioned properties, we propose approximating underlying data distribution not with one, but a mixture of Gaussian densities. Deriving Fisher vector for a Gaussian Mixture Model (GMM), reveals that batch normalization can be improved by independently normalizing with respect to the statistics of disentangled sub-populations. We refer to our proposed soft piecewise version of batch normalization as Mixture Normalization (MN). Through extensive set of experiments on CIFAR-10 and CIFAR-100, using both a 5-layers deep CNN and modern Inception-V3 architecture, we show that mixture normalization reduces required number of gradient updates to reach the maximum test accuracy of the batch normalized model by ~31%-47% across a variety of training scenarios. Replacing even a few BN modules with MN in the 48-layers deep Inception-V3 architecture is sufficient to not only obtain considerable training acceleration but also better final test accuracy. We show that similar observations are valid for 40 and 100-layers deep DenseNet architectures as well. We complement our study by evaluating the application of mixture normalization to the Generative Adversarial Networks (GANs), where “mode collapse” hinders the training process. We solely replace a few batch normalization layers in the generator with our proposed mixture normalization. Our experiments using Deep Convolutional GAN (DCGAN) on CIFAR-10 show that mixture normalized DCGAN not only provides an acceleration of ~58% but also reaches lower (better) “Fréchet Inception Distance” (FID) of 33.35 compared to 37.56 of its batch normalized counterpart.

**Index Terms**—Batch Normalization, Convolutional Neural Networks, Generative Probability Models, Gaussian Mixture Model, Fisher Vector.

## 1 INTRODUCTION

In the context of deep neural networks, the distribution of inputs to each layer is not only heavily dependent on the previous layers but it also changes as the network evolves through the training procedure. Ioffe and Szegedy [1] referred to this phenomenon as *internal covariate shift*. Another source of variation comes from the family of optimization techniques, specifically Stochastic Gradient Descent (SGD), that is used to train the deep neural networks. Although we can normalize a mini-batch at the input to the network, it will not remain normalized after multiple rounds of non-linear operations as we proceed from early layers to deeper ones. Batch Normalization (BN) [1] was proposed to alleviate the *internal covariate shift* by normalizing layer outputs for each training mini-batch with respect to its very own statistics, specifically mean and variance.

Batch normalization has been successful in dramatically accelerating training procedure of deep neural networks and is now a standard component in almost all deep convo-

lutional architectures. In fact, it is hard to imagine the possibility of effectively training state-of-the-art architectures such as Inception [2] [3], Residual [4] and Densely connected [5] networks, with hundreds of layers, without employing batch normalization.

Batch normalization [1] and its few extensions [6] [7] [8] [9], follow a general form (ref. Equation 6) to normalize a mini-batch, yet differ in the construction of the population over which mini-batch statistics are computed. In this paper, we provide a fresh view on the aforementioned general form of normalization, demonstrating its strong relation to the natural kernels that arise from generative probability models. Specifically, we show that assuming samples within a mini-batch are from the same probability density function, the general form of normalization is identical to the Fisher vector of a Gaussian distribution. Jaakkola and Haussler [10] proved that given classification labels as latent variables, Fisher kernel is asymptotically never inferior to the maximum *a posteriori* (MAP) decision rule. Therefore, the general form of normalization used in BN and its extensions, not only naturally emerges from the probability density function that models the generative process of the underlying data distribution, but also improves the discrimination

• M. M. Kalayeh and M. Shah are with the Center for Research in Computer Vision (CRCV), Department of Computer Science, University of Central Florida, Orlando, FL, 32816.  
E-mails: [mahdi@eecs.ucf.edu](mailto:mahdi@eecs.ucf.edu), [shah@crcv.ucf.edu](mailto:shah@crcv.ucf.edu)

power of the representation<sup>1</sup>.

However, for BN [1] to truly enjoy these properties, its input activations should follow a Gaussian distribution. We argue that due to the rectifying non-linearities employed in deep neural networks, it is unlikely that such condition is fully satisfied. Specifically, it is hard to believe that, in expectation, a linear combination (convolution operation) of multiple (different channels) distributions with semi-infinite support (output of previous *e.g* ReLU [11]), with at least two modes of variation one for the rectified values mapped to zero and one for the positive values, results in a single Gaussian distribution. We visualize internal activations of both a shallow and very deep architecture (ref. Section 4), and observe that indeed corresponding activations very often illustrate asymmetric characteristic and are better approximated by a mixture model rather than a single Gaussian distribution. This observation builds the core of our work.

To equip batch normalization with characteristics which Fisher kernel promises, we must first properly approximate the probability density function of the internal representations. From [12], we know that any continuous distribution can be approximated with arbitrary precision using a Gaussian Mixture Model (GMM). Hence, instead of computing one set of statistical measures from the entire population (of instances in the mini-batch) as BN does, we propose normalization on sub-populations which can be identified by disentangling modes of the distribution, estimated via GMM. We refer to our proposed technique as Mixture Normalization (MN).

While BN can only scale and/or shift the *whole* underlying probability density function, mixture normalization operates like a soft piecewise normalizing transform, capable of completely re-structuring the data distribution by independently scaling and/or shifting individual modes of distribution (ref. Figure 1).

We show that mixture normalization can effectively accelerate its batch normalization [13] counterpart, through reducing required number of gradient updates to reach the maximum test accuracy of the batch normalized model by **~31%-47%**, across a variety of training scenarios on CIFAR-10 and CIFAR-100. Mixture normalization handles training in large learning rate regime considerably better than batch normalization, and in majority of cases reaches even a better final test accuracy. It is worth pointing out that such acceleration in training procedure is obtained by solely replacing *a few* batch normalization modules with mixture normalization along the depth of the architecture. This is true both for shallow and very deep and modern architectures such as Inception-V3 [2] and DenseNet [5].

In summary, the main contributions of this work are as follows:

- We demonstrate how normalizing transform, employed in batch normalization and its variants, is related to the kernels from generative probability models.
- We show that, the distribution of activations associated with internal layers of deep convolutional neu-

1. This is established on the basis of the discriminative derivations of Fisher kernel (ref. Theorem 1 of [10]). The mathematical derivations can be found in Appendix A of the longer version of [10], available at <https://people.csail.mit.edu/tommi/papers/gendisc.ps>.

ral networks, illustrates an asymmetric characteristic and is very likely to be better estimated using a mixture model.

- We propose Mixture Normalization (MN), where a Gaussian mixture model initially identifies modes of variation in the underlying distribution, and then each sample in the mini-batch is normalized using mean and standard deviation of the mixture component to which it belongs to.
- We confirm through extensive set of experiments on CIFAR-10 and CIFAR-100, that Mixture Normalization not only significantly accelerates convergence of batch normalized models but also achieves better final test accuracy.

The remainder of this paper is organized as follows. Section 2 offers a detailed review of batch normalization and its various extensions. We then demonstrate the relationship between batch normalization and kernels from generative probability models in Section 3.1. This lays the groundwork for presenting our proposed mixture normalization in Section 3.2. Experimental results are shown in Section 4. Section 5 provides computational complexity analysis along with multiple detailed discussions attempting at an in-depth behavioral analysis of the mixture normalization. Finally, we conclude the paper in Section 6.

## 2 RELATED WORK

### 2.1 Batch Normalization

Let's consider  $x \in \mathbb{R}^{N \times C \times H \times W}$ , a 4-D activation tensor in a convolutional neural network where  $N$ ,  $C$ ,  $H$  and  $W$  are respectively the batch, channel, height and width axes. BN [1] computes the mini-batch mean ( $\mu_B$ ) and standard deviation ( $\sigma_B$ ), formulated in Equation 1, over the set  $\mathcal{B} = \{x_{1\dots m} : m \in [1, N] \times [1, H] \times [1, W]\}$ , where  $x$  is flattened across all but channel axis, and  $\epsilon$  is used for numerical stability.

$$\mu_B = \frac{1}{m} \sum_{i=1}^m x_i \quad \sigma_B = \sqrt{\frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2 + \epsilon} \quad (1)$$

If we assume that samples within the mini-batch are from the same distribution, the transform  $x \rightarrow \hat{x}$  shown in Equation 2, generates a zero mean and unit variance distribution. Then, BN uses learnable scale ( $\gamma$ ) and shift ( $\beta$ ) parameters to transform the normalized distribution into one with  $\beta$  mean and  $\gamma$  standard deviation.

$$\hat{x}_i = \frac{1}{\sigma_B} (x_i - \mu_B) \quad y_i = \gamma \hat{x}_i + \beta \quad (2)$$

Following Ioffe and Szegedy's [1] terminology, we refer to the transform

$$BN_{\gamma, \beta} : x_{1\dots m} \rightarrow y_{1\dots m} \quad (3)$$

as the *Batch Normalizing Transform*. In most of its applications, BN normalizes the output of a convolution layer just before the non-linear activation function (*e.g* ReLU [11]), which separates two consecutive convolution layers. As we discussed before, normalized activations,  $\hat{x}_{1\dots m}$ , under the

assumption that  $x_{1\dots m}$  are from the same distribution, are of zero mean and unit variance. Hence, applying ReLU function to  $\hat{x}_{1\dots m}$  can be approximately seen as rectifying half of the distribution. This may not necessarily be the optimum case from the perspective of minimizing the objective function during training. That is why the *learnable* scale ( $\gamma$ ) and more crucially shift ( $\beta$ ) parameter are important, as they allow the training procedure to shape the behavior of the non-linearity and consequently the entire model. It is easy to see (ref. Equation 4) that BN enables the the model to alternate between two extreme cases of ignoring the non-linearity and completely clipping the activations. While in the former, the effective depth of the network reduces as two back-to-back linear operations can be seen as one, the latter prevents the activations from propagating to the next layer.

$$\begin{aligned} \{\forall \gamma \exists \eta \lim_{\beta \rightarrow +\eta} \text{ReLU}(\text{BN}_{\gamma, \beta}(x)) = \text{BN}_{\gamma, \beta}(x) | 0 < \gamma, \eta \ll \infty\}, \\ \{\forall \gamma \exists \eta \lim_{\beta \rightarrow -\eta} \text{ReLU}(\text{BN}_{\gamma, \beta}(x)) = 0 | 0 < \gamma, \eta \ll \infty\}. \end{aligned} \quad (4)$$

The behavior of BN at inference is slightly different from training phase. To cope with potential discrepancy between distributions and dependency of each normalized activation to other instances in the mini-batch, BN accumulates a running average of the statistics at the training phase and uses them for normalizing mini-batches at inference. Hence, unlike training, where each mini-batch is normalized with respect to its very own statistics, all the mini-batches at inference use the same running average statistics for normalization while the scale and shift parameters are frozen. This workaround is effective when the size of the mini-batch is large, its instances are i.i.d. samples from training distribution and pre-computed statistics do not change [13]. However, in the absence of these conditions, estimation of mean and variance becomes less accurate at mini-batch level, hence affecting the running average statistics, and consequently results in performance degradation. To address these drawbacks, Ioffe [13] proposed *Batch Renormalization* where a per-dimension affine transformation is applied to the normalized activations as

$$\frac{x_i - \mu}{\sigma} = \frac{x_i - \mu_B}{\sigma_B} \cdot r + d, \text{ where } r = \frac{\sigma_B}{\sigma}, d = \frac{\mu_B - \mu}{\sigma}. \quad (5)$$

Note that the parameters of the transformation,  $r$  and  $d$ , are not trainable, instead they, in expectation, compensate for the difference between per mini-batch and over-population statistics. If the expected values of the per mini-batch statistics match the moving average ones, then the affine transformation reduces to an Identity resulting in batch renormalization [13] to behave identical to the batch normalization [1].

Large batch training of the neural networks is very well motivated as it reduces the training time by effectively leveraging the parallel and/or distributed computation. Yet the common practices and previous empirical observations [14] used to suggest that large batch training would not generalize as well as small batch training. Recently, Hoffer *et. al* [15] showed that there is no inherent generalization gap associated with large batch training once the number of iterations and learning rate are properly adapted. However,

they noticed that the dependency of statistics used by batch normalization [1], to the entire mini-batch, can affect the generalization of large batch training. To ameliorate that, Ghost Batch Normalization [15] was proposed, where a large batch is first scattered over multiple small virtual ("ghost") batches. Then, each sample is normalized with respect to the mean and standard deviation of the ghost batch to which it belongs to. At inference, normalization is done via a weighted running average that aggregates the statistics of all ghost batches during training.

Similarly, concerned with the dependency of batch-normalized activations on the entire mini-batch, Salimans *et. al* [16] proposed Weight Normalization, which operates on the weight vectors instead of activations. Hence, it does not introduce any dependencies between the samples within a mini-batch. At core, it decouples the norm ( $\mathbf{g}$ ) from the direction ( $\frac{\mathbf{v}}{\|\mathbf{v}\|}$ ) of the weight vectors. Specifically, considering  $\mathbf{w}$  as a weight vector in a standard artificial neural network, weigh normalization performs optimization over  $\mathbf{g}$  and  $\mathbf{v}$  using  $\mathbf{w} = \frac{\mathbf{g}}{\|\mathbf{v}\|} \mathbf{v}$  reparameterization. Salimans *et. al* [16] present weight normalization as a cheaper and less noisy approximation to the batch normalization [1] because first, convolutional neural networks usually have much fewer weights than activations and second, norm of  $\mathbf{v}$  is non-stochastic but mini-batch statistics can have high variance for small batch sizes. Unfortunately, the guarantees that [16] provides on the activations and gradients do not extend to models with arbitrary non-linearities or when the architecture contains layers without weight normalization [13].

## 2.2 Batch Normalization: Extended

Recently, a few extensions on batch normalization have been proposed, specifically, Layer Normalization (LN) [6], Instance Normalization (IN) [7], Group Normalization (GN) [8], and Divisive Normalization (DN) [9]. In this section, we loosely adopt notations from [9] and [8] to show what distinguishes all the aforementioned methods is solely the set on which sample statistics are computed. This unifying view was initially presented by Ren *et al.* [9] in an attempt to describe the relationship between BN, LN, and DN. Here we further extend it to IN and GN as well.

Let's consider  $i = (i_N, i_C, i_L)$  as a vector indexing the tensor of activations  $x \in \mathbb{R}^{N \times C \times L}$  associated to a convolution layer, where the spatial domain has been flattened ( $L = H \times W$ ). Then the general normalization,  $x \rightarrow \hat{x}$ , is in form of

$$v_i = x_i - \mathbb{E}_{\mathcal{B}_i}[x], \quad \hat{x}_i = \frac{v_i}{\sqrt{\mathbb{E}_{\mathcal{B}_i}[v^2] + \epsilon}}, \quad (6)$$

while similar to BN,  $\gamma$  and  $\beta$  parameters can be further applied to the normalized activations. Before describing the extensions to the batch normalization using Equation 6, it is worth pointing out that given

$$\mathcal{B}_i = \{j : j_N \in [1, N], j_C \in [i_C], j_L \in [1, L]\}, \quad (7)$$

the general normalization form reduces to the original batch normalization formulated in Equations 1 and 2.

**Layer Normalization (LN)** was proposed by Ba *et al.* [6], where they remove the inter-dependency of batch-normalized activations to the entire mini-batch. Despite its

effectiveness in recurrent networks, LN underperforms BN when applied to the convolution layers. That is because LN enforces the same distribution on the entire spatial domain *and* along channel axis which is not natural in case of convolution layers as the visual information can dramatically vary over the spatial domain. LN can be formulated as Equation 6 when

$$\mathcal{B}_i = \{j : j_N \in [i_N], j_C \in [1, C], j_L \in [1, L]\}. \quad (8)$$

**Instance Normalization (IN)** was proposed by Ulyanov *et al.* [7] for the problem of image style transfer [17]. While enjoying no inter-dependency to other samples in the mini-batch, IN [7] uses more relaxed conditions than LN [6], by computing the statistics only over the spatial domain generating different mean and standard deviations for each sample *and* each channel. IN can be formulated as Equation 6 when

$$\mathcal{B}_i = \{j : j_N \in [i_N], j_C \in [i_C], j_L \in [1, L]\}. \quad (9)$$

**Group Normalization (GN)** [8] is somewhere in between LN and IN. GN divides the channels into multiple groups ( $G = 32$  by default), then computes the statistics along  $L$  axis but only within a subgroup of the channels. Therefore, when the number of groups matches the channel size ( $G = C$ ), GN is identical to IN. On the other hand, when there is only one group ( $G = 1$ ), GN reduces to LN. GN has shown to be effective on image classification, object detection and segmentation, when batch size is very small (2 and 4) while providing comparably good results with BN on typical batch sizes. GN can be formulated as Equation 6 when

$$\mathcal{B}_i = \{j : j_N \in [i_N], j_C \in [1, C], j_L \in [1, L] \mid \lfloor \frac{j_C}{C/G} \rfloor = \lfloor \frac{i_C}{C/G} \rfloor\}, \quad (10)$$

where  $\lfloor \frac{j_C}{C/G} \rfloor$  ensures that normalizing  $x_i$  is only influenced by the activation vectors which fall within the same group as  $x_i$ .

**Divisive Normalization (DN)** [9] can be seen as a local version of LN, where in normalizing  $x_i$ , instead of all the activation vectors within the same layer, only those that are in a certain vicinity of  $x_i$  contribute. This very well addresses the aforementioned drawback of LN, when applied to the convolution layer. DN is formulated slightly different from the aforementioned normalization methods, specifically as

$$v_i = x_i - \mathbb{E}_{\mathcal{A}_i}[x] \quad \hat{x}_i = \frac{v_i}{\sqrt{\rho^2 + \mathbb{E}_{\mathcal{B}_i}[v^2]}}, \quad (11)$$

when

$$\mathcal{A}_i = \{j : d(x_i, x_j) \leq R_{\mathcal{A}}\} \quad \mathcal{B}_i = \{j : d(v_i, v_j) \leq R_{\mathcal{B}}\}, \quad (12)$$

where  $d$  denotes an arbitrary distance between two hidden units,  $\rho$  is the normalizer bias, and  $R$  denotes the neighbourhood radius. Ren *et al.* [9] have shown how varying  $\rho$  would allow DN followed by ReLU to alternate within a wide range of non-linear behaviors. DN shows promising results both on convolutional and recurrent networks and can be easily implemented as a convolutional operator where  $R_{\mathcal{A}}$  and  $R_{\mathcal{B}}$  are determined by the kernel size.

### 3 PROPOSED METHOD

In this section, we show how the formulation of batch normalization and its extensions is related to the kernels from generative probability models, and more specifically Fisher kernel. Then, we explain why in the context of deep neural networks, due to the non-linearities, distribution of the activations is almost certainly of multiples modes of variation. Finally, we detail our proposed mixture normalization, where we employ Gaussian mixture model to identify sub-populations and normalize with respect to not one, but multiple components that comprise the data distribution.

#### 3.1 Kernels from Generative Probability Models

So far, we've described variations of batch normalization in an unifying framework and shown what distinguishes them is the set of samples over which certain statistics, specifically mean and standard deviation are computed. In this section, we revisit Fisher kernel, the seminal work of Jaakkola and Haussler [10], in order to demonstrate how the general form of normalization, specified in Equation 6, is related to the kernels that naturally arise from the generative probability models.

**Fisher Kernel:** Let  $\mathcal{B} = \{x_{1 \dots m}\}$ , regardless of how the tensor of activations has been indexed (ref. Section 2.2), be a set of  $m$  observations  $x_i \in \mathcal{X}$ , associated to a sample mini-batch, and  $\mathcal{P} = \{p_{\theta}, \theta \in \Theta\}$  be a suitably regular parametric family of distributions, where  $p_{\theta}$  models the generative process of samples in  $\mathcal{X}$ <sup>2</sup>. Based on the theory of information geometry [18],  $\mathcal{P}$  defines a Riemannian manifold  $M_{\Theta}$ , with a local metric given by the Fisher Information Matrix

$$F_{\theta} = \mathbb{E}_{x \sim p_{\theta}}[G_{\theta}^{\mathcal{B}} G_{\theta}^{\mathcal{B}^T}], \quad (13)$$

where

$$G_{\theta}^{\mathcal{B}} = \nabla_{\theta} \log p_{\theta}(\mathcal{B}), \quad (14)$$

the gradient of the log-likelihood of  $p_{\theta}$  at  $\mathcal{B}$ , also known as *Fisher score*, determines the direction of steepest ascent in  $\log p_{\theta}(\mathcal{B})$  along the manifold. In other words,  $G_{\theta}^{\mathcal{B}}$  describes modifications which the current model parameters  $\theta$  need in order to maximize  $\log p_{\theta}(\mathcal{B})$  [10]. The natural gradient [19] is then defined as

$$\phi_{\mathcal{B}} = F_{\theta}^{-1} G_{\theta}^{\mathcal{B}}, \quad (15)$$

based on which [10] proposes natural kernel, the inner product between natural gradients relative to the local Riemannian metric:

$$K(\mathcal{B}_j, \mathcal{B}_i) \propto \phi_{\mathcal{B}_j}^T F_{\theta} \phi_{\mathcal{B}_i} = G_{\theta}^{\mathcal{B}_j^T} F_{\theta}^{-1} G_{\theta}^{\mathcal{B}_i}. \quad (16)$$

Jaakkola and Haussler [10] refer to this as *Fisher kernel* and prove that given classification labels as latent variable, Fisher kernel is asymptotically never inferior to the maximum *a posteriori* (MAP) decision rule (ref. Theorem 1 in [10]). Hence, we arrive at a similarity measure which is naturally induced from the probability density function that models the generative process of  $\mathcal{X}$  and simultaneously improves the discrimination power of the model.

2. In other words,  $\mathcal{X}$  is an unknown generative process which is modeled by  $p_{\theta}$ , and we assume that instances in a mini-batch are sampled from it.

Sanchez *et. al* [20] have beautifully pointed out that since  $F_\theta$  is positive semi-definite, its inverse has a Cholesky decomposition  $F_\theta^{-1} = L_\theta^T L_\theta$ , which allows re-writing Equation 16 as

$$K(\mathcal{B}_j, \mathcal{B}_i) = \mathcal{G}_\theta^{\mathcal{B}_j^T} \mathcal{G}_\theta^{\mathcal{B}_i}, \quad (17)$$

where

$$\mathcal{G}_\theta^{\mathcal{B}} = L_\theta \nabla_\theta \log p_\theta(\mathcal{B}), \quad (18)$$

the normalized gradient vector of  $\mathcal{B}$ , is known as *Fisher vector* [20]. Such explicit transformation  $\mathcal{B} \rightarrow \mathcal{G}_\theta^{\mathcal{B}}$  enjoys all the interesting characteristics of Fisher kernel [10], while being tailored for linear operations.

**Fisher Vector for Gaussian Distribution:** In order to derive Fisher kernel [10], we noted that  $p_\theta$  only needs to be of a family of regular parametric distributions. However, we have not yet parameterized it. Let's consider  $p_\theta$  to be modelled by Gaussian distribution

$$p_\theta(x) = \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}} \exp \left\{ -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right\}, \quad (19)$$

where  $x \in \mathbb{R}^D$  and  $\theta = \{\mu, \Sigma\}$  is the set of model parameters. The gradients of the log-likelihood of  $p_\theta(x)$  with respect to  $\mu$  and  $\Sigma$  are then formulated as

$$\begin{aligned} \nabla_\mu \log p_\theta(x) &= -\frac{1}{2} \left( \frac{\partial (x - \mu)^T \Sigma^{-1} (x - \mu)}{\partial \mu} \right) = \Sigma^{-1} (x - \mu), \\ \nabla_\Sigma \log p_\theta(x) &= -\frac{1}{2} \left( \frac{\partial \log(|\Sigma|)}{\partial \Sigma} + \frac{\partial (x - \mu)^T \Sigma^{-1} (x - \mu)}{\partial \Sigma} \right), \\ &= -\frac{1}{2} \left( \Sigma^{-1} - \Sigma^{-1} (x - \mu) (x - \mu)^T \Sigma^{-1} \right). \end{aligned} \quad (20)$$

If we assume  $\Sigma$  to be diagonal <sup>3</sup> with  $\Sigma_{i,i} = \sigma_{i,i}^2$ , we can re-write the gradients as

$$\begin{aligned} \nabla_\mu \log p_\theta(x) &= \frac{1}{\sigma^2} (x - \mu), \\ \nabla_\sigma \log p_\theta(x) &= -\frac{1}{\sigma} + \frac{1}{\sigma^3} (x - \mu)^2. \end{aligned} \quad (21)$$

Substituting Equation 21 in Equation 14 results in

$$G_\theta^x = \begin{bmatrix} \nabla_\mu \log p_\theta(x) \\ \nabla_\sigma \log p_\theta(x) \end{bmatrix} = \begin{bmatrix} \frac{1}{\sigma^2} (x - \mu) \\ -\frac{1}{\sigma} + \frac{1}{\sigma^3} (x - \mu)^2 \end{bmatrix}. \quad (22)$$

We then use non-central moments of univariate normal distribution to calculate Equation 13 given Equation 22, arriving at

$$F_\theta = \begin{bmatrix} \frac{1}{\sigma^2} & 0 \\ 0 & \frac{2}{\sigma^2} \end{bmatrix}, \quad (23)$$

3. Such assumption is motivated by the fact that computing full covariance matrix and its inverse makes both the normalization and the back-propagation calculations very expensive [1].

the Fisher information matrix of Gaussian distribution. Finally, the Fisher vector for Gaussian distribution (ref. Equation 18) can be characterized as

$$\begin{aligned} \mathcal{G}_\theta^x &= \begin{bmatrix} \mathcal{G}_\mu^x \\ \mathcal{G}_\sigma^x \end{bmatrix}, \\ &= \begin{bmatrix} \sigma & 0 \\ 0 & \frac{1}{\sqrt{2}}\sigma \end{bmatrix} \begin{bmatrix} \frac{1}{\sigma^2} (x - \mu) \\ -\frac{1}{\sigma} + \frac{1}{\sigma^3} (x - \mu)^2 \end{bmatrix}, \\ &= \begin{bmatrix} \frac{1}{\sigma} (x - \mu) \\ \frac{1}{\sqrt{2}} \left( -1 + \frac{(x - \mu)^2}{\sigma^2} \right) \end{bmatrix}. \end{aligned} \quad (24)$$

We observe that the general form of normalization, formulated in Equation 6, has in fact emerged in  $\mathcal{G}_\mu^x$ . Hence, we have shown that indeed batch normalization and its various extensions are closely related to the natural kernel that arises from generative probability model, describing the underlying data distribution <sup>4</sup>.

### 3.2 Mixture Normalization

We believe that in the context of deep neural networks, due to non-linear activation functions, distribution of layer outputs are very likely asymmetric. That is, the hypothesis which a Gaussian distribution can model  $p_\theta$  is less likely to be valid. Therefore, to properly approximate the probability density function, we propose employing generative *mixture* models. Consequently, instead of computing one set of statistical measures from the entire population, we propose to frame batch normalizing transform on sub-populations which can be identified by disentangling modes of variation.

**Intuition:** Let's consider  $x_l \in \mathbb{R}^{N \times C_l \times H_l \times W_l}$  to be the input activation tensor to the  $l^{th}$  layer, denoted by  $\omega_l$ , in a convolutional neural network. Again,  $N$ ,  $C_l$ ,  $H_l$  and  $W_l$  are respectively batch, channel, height and width axes at the corresponding layer. Batch normalization layers are indexed similarly. Since the batch size is fixed throughout the network, we drop the subscript  $l$  from  $N_l$ . For the sake of simplicity, we ignore pooling layers and assume the non-linearity activation functions to be rectified linear units (ReLU) [11], since pooling layers can be trivially added to the formulation. Similarly, other non-linearity functions can partially or fully replace ReLU throughout the network. Given these,

$$\begin{aligned} x_{l-1} &= \text{ReLU}(\text{BN}_{l-2}(x_{l-2} * \omega_{l-2})), \\ x_l &= \text{ReLU}(\text{BN}_{l-1}(x_{l-1} * \omega_{l-1})), \\ x_{l+1} &= \text{ReLU}(\text{BN}_l(x_l * \omega_l)), \end{aligned} \quad (25)$$

formulate three consecutive layers in the aforementioned convolution neural network where  $*$  represents the convolution operation. In Section 3.1, given the hypothesis  $x \sim \mathcal{N}(\mu, \sigma^2)$ , the general form of normalization in Equation 6 emerges as the Fisher vector with respect to the mean of the distribution, namely  $\mathcal{G}_\mu^x$ . This means that the

4. Based on our derivations, we "expect" that integrating  $\mathcal{G}_\mu^x$  in addition to  $\mathcal{G}_\sigma^x$  further improves the normalization. Yet, at the time, we do not have experimental results to support this.

input to  $BN_l$  at the  $l^{th}$  layer in Equation 25, specifically  $x_l * \omega_l$ , is of a Gaussian distribution. Since, convolution is a linear operation, and Gaussian distribution is closed under linear combination, it further implies that  $x_l \sim \mathcal{N}(\mu_l, \sigma_l^2)$ . However,  $x_l$  is the output of ReLU from the  $(l-1)^{th}$  layer with semi-infinite support on  $[0, +\infty)$ . Hence, assuming that a symmetric probability density function, like Gaussian distribution, with support on the whole Real line can model the linear combination of outputs of multiple rectified linear units, is not very well justified.

Based on this observation, we propose to parameterize the probability density function,  $p_\theta$  (ref. Section 3.1), as a Gaussian Mixture Model (GMM). From [12], we know that any continuous distribution can be approximated with arbitrary precision using a GMM. Since our input distribution is generated by linear combination (through convolution operation) of rectified Gaussian distributions, it is not necessarily continuous and contains, at least two modes, one for the rectified values mapped to zero and one for the positive values. Therefore, we expect a mixture model to provide us with a better approximation of such distribution than a single Gaussian model.

**Fisher Vector for Gaussian Mixture Model:** In the following, to prevent clutter, we drop subscript  $\theta$  where  $\theta = \{\lambda_k, \mu_k, \Sigma_k : k = 1, \dots, K\}$ . Then, for  $x \in \mathbb{R}^D$ , without loss of generality, consider

$$p(x) = \sum_{k=1}^K \lambda_k p_k(x) \quad s.t. \forall_k : \lambda_k \geq 0, \sum_{k=1}^K \lambda_k = 1, \quad (26)$$

where

$$p_k(x) = \frac{1}{(2\pi)^{D/2} |\Sigma_k|^{1/2}} \exp \left\{ -\frac{1}{2} (x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) \right\}, \quad (27)$$

represents  $k^{th}$  Gaussian in the mixture model  $p(x)$ . To remove the probability simplex constraint on  $\lambda$  in Equation 26, a commonly used change of variables [21] is employed as

$$\alpha_k = \log \left( \frac{\lambda_k}{\lambda_K} \right), \quad k = 1, \dots, K. \quad (28)$$

Assume  $\alpha_K = 0$  to be constant, then we can re-write Equation 26 as

$$p(x) = \sum_{k=1}^K \left( \frac{\exp(\alpha_k)}{\sum_{j=1}^K \exp(\alpha_j)} \right) p_k(x). \quad (29)$$

Following similar derivations as in Section 3.1, given diagonal covariance assumption, the gradients of the log-likelihood of  $p(x)$  with respect to  $\mu_k$  and  $\sigma_k$  can be written as

$$\begin{aligned} \nabla_{\mu_k} \log p(x) &= \nu_k(x) \left( \frac{x - \mu_k}{\sigma_k^2} \right), \\ \nabla_{\sigma_k} \log p(x) &= \nu_k(x) \left( -\frac{1}{\sigma_k} + \frac{(x - \mu_k)^2}{\sigma_k^3} \right). \end{aligned} \quad (30)$$

where

$$\nu_k(x) = \frac{\lambda_k p_k(x)}{\sum_{j=1}^K \lambda_j p_j(x)}, \quad (31)$$

is the probability that  $x$  has been generated by the  $k^{th}$  Gaussian component in the mixture model. Given soft assignment distribution in Equation 31, Perronnin *et. al* [22] proposed a closed form approximation to the Fisher information matrix of GMM<sup>5</sup> as

$$F_k = \begin{bmatrix} \frac{\lambda_k}{\sigma_k^2} & 0 \\ 0 & \frac{2\lambda_k}{\sigma_k^2} \end{bmatrix}, \quad (32)$$

resulting in

$$\begin{aligned} \mathcal{G}_k^x &= \begin{bmatrix} \mathcal{G}_{\mu_k}^x \\ \mathcal{G}_{\sigma_k}^x \end{bmatrix}, \\ &= \begin{bmatrix} \frac{\sigma_k}{\sqrt{\lambda_k}} & 0 \\ 0 & \frac{\sigma_k}{\sqrt{2\lambda_k}} \end{bmatrix} \begin{bmatrix} \nu_k(x) \left( \frac{x - \mu_k}{\sigma_k^2} \right) \\ \nu_k(x) \left( -\frac{1}{\sigma_k} + \frac{(x - \mu_k)^2}{\sigma_k^3} \right) \end{bmatrix}, \\ &= \frac{\nu_k(x)}{\sqrt{\lambda_k}} \begin{bmatrix} \frac{1}{\sigma_k} (x - \mu_k) \\ \frac{1}{\sqrt{2}} \left( -1 + \frac{(x - \mu_k)^2}{\sigma_k^2} \right) \end{bmatrix}. \end{aligned} \quad (33)$$

Comparing Equation 33 with Equation 24, we observe that Fisher vector for each component of a GMM is very similar to the one obtained from a single Gaussian distribution. However, there are two main differences:

- $\nu_k(x)$  in a soft-assignment mechanism, scales the  $k^{th}$  Fisher vector based on the posterior probability of  $x$  being generated by the corresponding Gaussian component.
- $\lambda_k$  normalizes the  $k^{th}$  Fisher vector based on the contribution of the  $k^{th}$  mixture component in approximating the underlying data distribution.

We have previously shown (ref. Section 3.1) that the normalization formulated in batch normalization and its extensions emerges as natural kernel, if we assume that the underlying data distribution can be modelled by a single Gaussian distribution. We then explained that due to non-linear activation functions, such hypothesis cannot properly address the characteristics of the distribution. We employed Gaussian mixture model as an alternative and showed that natural kernel induced from GMM, follows a similar normalization mechanism, except normalization is done with respect to multiple sets of statistics obtained from different sub-populations.

**Formulation:** Based on the aforementioned observations, we propose Mixture Normalization. Let's consider  $i = (i_N, i_C, i_L)$  as a vector indexing the tensor of activations  $x \in \mathbb{R}^{N \times C \times L}$  associated to a convolution layer, where the spatial domain has been flattened ( $L = H \times W$ ). Then the *Mixture Normalizing Transform* is defined as

$$\hat{x}_i = \sum_{k=1}^K \frac{\nu_k(x_i)}{\sqrt{\lambda_k}} \cdot \hat{x}_i^k, \quad (34)$$

5. For more detailed derivation of Fisher information matrix and Fisher vector in Gaussian mixture models, readers are encouraged to refer to [22] and [20].

given

$$v_i^k = x_i - \mathbb{E}_{\mathcal{B}_i}[\hat{\nu}_k(x) \cdot x], \quad \hat{x}_i^k = \frac{v_i^k}{\sqrt{\mathbb{E}_{\mathcal{B}_i}[\hat{\nu}_k(x) \cdot (v^k)^2] + \epsilon}}, \quad (35)$$

where

$$\hat{\nu}_k(x_i) = \frac{\nu_k(x_i)}{\sum_{j \in \mathcal{B}_i} \nu_k(x_j)}, \quad (36)$$

is the normalized contribution of  $x_i$  over  $\mathcal{B}_i$  in estimating statistical measures of the  $k^{th}$  Gaussian component. Similar to batch normalization, we can also include additional parameters by slightly modifying Equation 35 using

$$\begin{aligned} \mathbb{E}_{\mathcal{B}_i}[\hat{\nu}_k(x) \cdot x] &\rightarrow \mathbb{E}_{\mathcal{B}_i}[\hat{\nu}_k(x) \cdot x] + \beta, \\ \mathbb{E}_{\mathcal{B}_i}[\hat{\nu}_k(x) \cdot (v^k)^2] &\rightarrow \mathbb{E}_{\mathcal{B}_i}[\hat{\nu}_k(x) \cdot (v^k)^2] \cdot \gamma, \end{aligned} \quad (37)$$

where  $\beta$  and  $\gamma$  respectively indicate scale and shift. Our formulation is general (e.g.  $H, W = 1$  for fully-connected layers) and can be applied to all the variations of batch normalization, detailed in Section 2.2, simply by constructing  $\mathcal{B}_i$  accordingly.

During training, we fit a Gaussian mixture model to  $\mathcal{B} = \{x_{1\dots m} : m \in [1, N] \times [1, L]\}$  by Maximum Likelihood Estimation (MLE). This is a two-stage process where we use the seeding procedure of K-means++ [23] to initialize the centers of the mixture component. Then, the parameters of the mixture model,  $\theta = \{\lambda_k, \mu_k, \Sigma_k : k = 1, \dots, K\}$ , are estimated by Expectation-Maximization (EM) [24]. In practice, one or two EM iterations are sufficient, thanks to proper and efficient initialization by K-means++ [23]. We then normalize  $x_1, x_2, \dots, x_m$  with respect to the estimated parameters (Equations 35) and aggregate using posterior probabilities (Equations 34). At inference, following the batch normalization [1], we want to normalize a sample mini-batch with respect to the statistics of the training data. To do so, we can maintain last  $T$  mini-batches from the training stage. This can be simply implemented by a queue of length  $T$ . We aggregate all instances in the queue into a large pool of samples and fit a Gaussian mixture model to it. Estimated parameters are then used to normalize all the mini-batches of the test set. While being effective, the above strategy is not very desirable as it always maintains  $T$  mini-batch of samples in GPU memory. Yet, there is a very simple workaround to significantly improve this. Specifically, during training, instead of the samples, we only need to keep the estimated parameters resulting in the queue  $\{\theta_0, \theta_1 \dots \theta_{T-1}\}$ , where the subscript indicates the position in the queue. Note that in this case, queue always maintains  $(2KC+K) \times T$  values in the GPU memory which is orders of magnitude smaller than  $(mC) \times T$  of the previous strategy. The normalization will then be performed via

$$\hat{x}_i = \sum_{t=0}^{T-1} \sum_{k=1}^K \frac{1}{\sqrt{\tau^t \lambda_k^t}} \frac{\tau^t \lambda_k^t p_k^t(x_i)}{\sum_{q=0}^{T-1} \sum_{j=1}^K \tau^q \lambda_j^q p_j^q(x_i)} \left( \frac{x_i - \mu_k^t}{\sigma_k^t} \right), \quad (38)$$

where

$$\tau^t = \frac{(1 - \zeta)}{(1 - \zeta^T)} \zeta^{(T-t-1)} \quad (39)$$

is the decay factor normalized with respect to the sum of its geometric series. Here, estimated mixture models from last  $T$  mini-batch of training stage are aggregated with a scale proportional to their chronological order in the queue. By default,  $\zeta$  and  $T$  are respectively set to 0.9 and 10.

**Differentiability and Gradient Propagation:** Despite our end-to-end training, K-means++ [23] seeding procedure and EM iterations (to fit the Gaussian mixture model) are performed outside the computational graph of the neural network<sup>6</sup>. Hence, MN is not fully differentiable. In the following, we explain how gradient back-propagates through MN.

Let  $\nu \in \mathbb{R}^{m \times K}$  be a matrix where its  $(i, j)^{th}$  element represents the probability that  $x_i$ , the  $i^{th}$  row of  $\mathbf{x} \in \mathbb{R}^{m \times D}$  ( $\mathcal{B}$  in matrix format), has been generated by the  $j^{th}$  Gaussian component in the mixture model (Equation 31). The process of obtaining  $\nu$  is not differentialble because of K-means++ and following EM iterations, yet we can write MN equations such that with exception of  $\nu$ , it is fully differentiable.

Following Equation 36, we obtain the normalized contribution of each sample point  $x_i$  in estimating statistical measures of every Gaussian component via dividing each column of  $\nu$  by the sum of its elements. This results in  $\hat{\nu}$ , the matrix of responsibilities using which we calculate  $\mu = \hat{\nu}^T \mathbf{x}$  and  $\sigma^2 = \hat{\nu}^T \mathbf{x}^2 - \mu^2$  where  $\mu \in \mathbb{R}^{K \times D}$ ,  $\sigma^2 \in \mathbb{R}^{K \times D}$ , and  $\lambda_k$  is obtained by averaging the  $k^{th}$  column of  $\nu$ . As we can see, all these calculations are simple differentiable operations. Hence, while gradient stops at  $\nu$ , it seamlessly back-propagates through the rest of MN equations, relating parameters of the mixture component with samples and their posterior.

**Effect of Non-linearity:** When mixture normalization is followed by ReLU [11] or similar non-linearity activation functions, rectifying should be applied to the per-component normalized activations (i.e.  $\hat{x}_i^k$ ) as

$$\text{ReLU}(\text{MN}(x_i)) := \sum_{k=1}^K \frac{\nu_k(x_i)}{\sqrt{\lambda_k}} \cdot \text{ReLU}(\hat{x}_i^k), \quad (40)$$

where MN stands for mixture normalization. In practice, given  $K \geq 3$ ,  $\nu(x_i) = [\nu_1(x_i), \nu_2(x_i), \dots, \nu_K(x_i)]$  is mostly very sparse, meaning that only 1 out of  $K$  elements is considerably larger than zero. As a result, we have

$$\sum_{k=1}^K \frac{\nu_k(x_i)}{\sqrt{\lambda_k}} \cdot \text{ReLU}(\hat{x}_i^k) \approx \text{ReLU}\left(\sum_{k=1}^K \frac{\nu_k(x_i)}{\sqrt{\lambda_k}} \cdot \hat{x}_i^k\right), \quad (41)$$

a format which is the same as how ReLU follows batch normalization. However, when Gaussian components considerably overlap, the aforementioned approximation is less accurate.

**Local vs. Global Normalization:** Batch normalization is an Affine transform on the *whole* probability density function of the underlying distribution. In other words, *all* the samples from the distribution are normalized using the *same* mean and standard deviation, estimated from the entire population. In contrast, in proposed mixture normalization, we transform each sample using mean and standard

6. In future, we are going to explore mixture normalization, when estimating GMM via batch and stochastic Riemannian optimization [25].

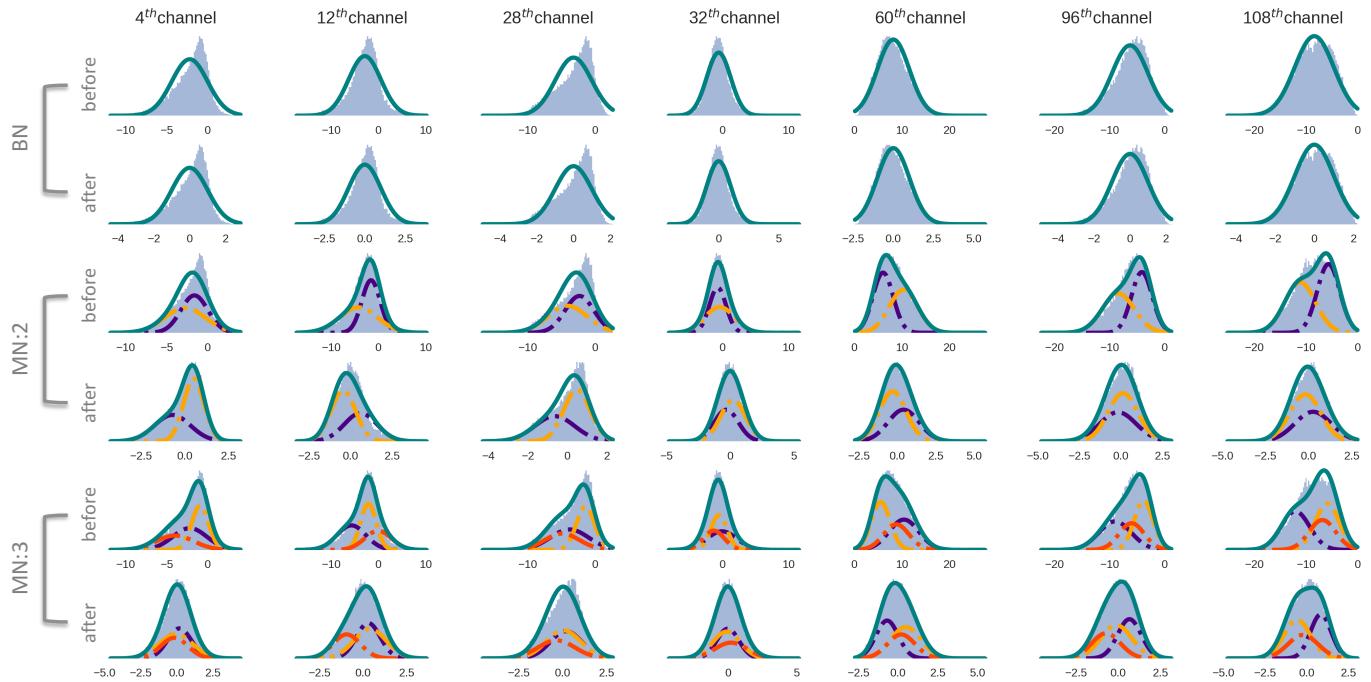


Fig. 1: Visualizing Mixture Normalization: Given a random mini-batch in the midway of training on CIFAR-100, we illustrate the underlying distribution of activations (output of convolution) associated with a random subset of 128 channels in the layer “conv2” of CIFAR CNN architecture (detailed in Table 1). Solid teal curve indicates the probability density function. Dashed curves represent different mixture components shown in various colors. Note that similar colors across multiple subfigures, simply index mixture components and do not indicate any association. We observe that mixture normalization, shown by MN:2 and MN:3 (2 and 3 respectively represent the number of components in the mixture of Gaussians), provides better approximation,  $p(x)$ , illustrated by solid teal curve, to the underlying distribution. Also, mixture normalized activations, in comparison to the batch normalized ones, are considerably closer to the normal distribution and illustrate less skewed probability densities.

deviation of the mixture component to which the sample belongs to. Therefore, one can see mixture normalization as a soft piecewise normalizing transform. Figure 1 illustrates the aforementioned differences. Note how in case of batch normalization,  $p(x)$ , modeled by a Gaussian density function, fails to properly approximate the underlying data distribution. This problem is more severe when the distribution is skewed. Mixture normalization instead, is capable of handling these challenges as it disentangles the modes of variation. In Figure 1, we show mixture normalization with  $K$  set to 2 and 3. Obviously, when the number of components is set to 1, mixture normalization reduces to the general form of normalization. We observe that even with two components, mixture normalization provides a better approximation to the underlying data distribution. However, in some cases (e.g. columns correspond to the 4<sup>th</sup> and 28<sup>th</sup> channels), we can benefit from increasing the number of mixture components. Mixture normalized activations, in comparison to the batch normalized ones, are considerably closer to the normal distribution, and illustrate less skewed probability densities.

## 4 EXPERIMENTS

To evaluate the effectiveness of our proposed mixture normalization, we conduct extensive set of experiments on CIFAR-10 and CIFAR-100 datasets [26]. We compare mixture

normalization against its batch normalization counterpart in a variety of settings given four backbone choices, namely a 5-layers deep fully convolutional neural network, the Inception-V3 [2], along with the 40-layers and 100-layers deep DenseNet [5] architectures. According to the literature [8] [9] [13], when mini-batch size is sufficiently large (e.g 32), standard batch normalization [1] outperforms its variants (ref. Section 2.2) in image classification. Although, this is not valid in recurrent networks. Therefore, here, we only compare against standard batch normalization [1].

It is important to emphasize, that we do not aim at achieving state-of-the-art results as it requires employing computationally expensive architectures, and involves careful tuning of many hyper-parameters and heuristics. Instead, we are interested in understanding the behavior of mixture normalization. We focus on demonstrating that solely replacing *a few* batch normalization with our proposed mixture normalization, can dramatically increase the convergence rate, and in majority of cases even yields a better final test accuracy.

In summary, in this section:

- We compare BN and MN, in small and large learning rate regimes, using both shallow and very deep CNN architectures.
- We show the effect of applying MN to different layers, while varying the number of mixture com-

ponents and EM iterations.

- We quantify the acceleration obtained using MN, by measuring required number of gradient updates for mixture normalized model, in order to reach the best test accuracy of its batch normalized counterpart.
- We demonstrate, that our findings are consistent with respect to the choice of the optimization technique (adaptive v.s non-adaptive) and learning rate decay policies.
- We finish this section by demonstrating the application of mixture normalization in Generative adversarial networks (GANs) [27].

#### 4.1 Datasets

For our experiments, we use popular CIFAR [26] datasets. Images are  $32 \times 32$  and labeled with 10 and 100 classes, respectively for CIFAR-10 and CIFAR-100. We follow the standard data split where 50K images are used for training and 10K for testing. As for preprocessing, we normalize images with respect to the mean and standard deviation of the dataset. Also, similar to the previous works [4] [5] [28] [29] [30] on CIFAR, we adopt horizontal flipping and random cropping for data augmentation.

#### 4.2 CIFAR CNN

We begin with a small 5-layers deep convolutional neural network architecture detailed in Table 1, where we later replace batch normalization in certain layers with mixture normalization. Following [1], we experiment using two different learning rates, one 5 times larger than the other. Additionally, we do the same with the weight decay to very well cover various training regimes. We use RMSprop [31] with 0.9 momentum and exponentially reduce the initial learning rate every two epochs with the decay rate of 0.93. The size of mini-batch is set to 256 and we train all the models for 100 epochs. To facilitate comparison between different training settings, we index experiments accordingly, where BN and MN, respectively, indicate usage of batch and mixture normalization.

layer	type	size	kernel	(stride, pad)
input	input	$3 \times 32 \times 32$	–	–
conv1	conv+bn+relu	$64 \times 32 \times 32$	$5 \times 5$	(1, 2)
pool1	max pool	$64 \times 16 \times 16$	$3 \times 3$	(2, 0)
conv2	conv+bn+relu	$128 \times 16 \times 16$	$5 \times 5$	(1, 2)
pool2	max pool	$128 \times 8 \times 8$	$3 \times 3$	(2, 0)
conv3	conv+bn+relu	$128 \times 8 \times 8$	$5 \times 5$	(1, 2)
pool3	max pool	$128 \times 4 \times 4$	$3 \times 3$	(2, 0)
conv4	conv+bn+relu	$256 \times 4 \times 4$	$3 \times 3$	(1, 1)
pool4	avg pool	$256 \times 1 \times 1$	$4 \times 4$	(1, 0)
linear	linear	10 or 100	–	–

TABLE 1: CIFAR CNN architecture

In the first set of experiments, for different mixture normalization variations, shown in Table 2 by MN-1 to MN-4, we replace the batch normalization in the “conv3” layer (ref. Table 1) with mixture normalization while keeping the remaining layers intact. From Table 2 and Figure 2, we observe that irrespective of the weight decay and learning rates, not only MN models converge much faster than their

corresponding BN counterparts, but they achieve a better test accuracy after 100 epochs.

**Large learning rate:** When we increase the learning rate from 0.01 to 0.05, the convergence gap is even larger (ref. Table 2 and Figure 2), demonstrating the capability of mixture normalization to better utilize larger learning rates for training. Note, that for the large learning rate regime to eventually match the final learning rate of small learning rate regime, either the exponential decay rate should be reduced (e.g. 0.91 instead of 0.93) or the number of times it is applied must be increased (e.g. every epoch instead of every 2 epochs). However, here we keep the same learning rate decay policy to probe the sole effect of increasing the learning rate.

**Stability:** Another observation is with regards to the uncertainty of the model predictions. In Figure 2, we show one standard deviation (shaded area) computed within a window of 10 epochs for all the test error curves. We can see that models trained with mixture normalization illustrate a considerably more stable test error, meaning that, from one epoch to another, it is less likely that the performance fluctuates abruptly. This alongside faster convergence [32] is reminiscent of relatively flat minima in the optimization landscape [33], where the classification margin of the model, as a whole, is robust with respect to small changes (gradient updates) to the model parameters. Note that in all these experiments, we have solely replaced one batch normalization layer with mixture normalization.

**Effect of parameters:** Solving for the parameters of GMM begins with an initialization using K-means clustering. To obtain the best initial seeds, we use  $2 + \log(K)$  trials and then perform standard K-means clustering for a certain number of iterations. The result is then used to initiate the GMM parameters, which is later followed by a fixed number of EM updates. In our experiments, we use the same number of iterations at two phases and report their summation as “EM iter.” in Table 2 (and later in Table 4). From Figure 3 (left) and Table 2, we observe that more EM iterations, provides faster convergence and better final test accuracy. This is aligned with our expectation as more EM iterations translates to better approximation of the underlying data distribution. However, the downside is that more EM iterations results in increasing the computation time. So far, we have only modified one layer in CIFAR CNN architecture. In Figure 3 (right), we illustrate that similar behavior can be observed when mixture normalization is employed at earlier layers like “conv2”. Furthermore, model enjoys even faster convergence rate in addition to better final test accuracy (ref. Table 2), when more layers (“conv2” and “conv3”) are equipped with mixture normalization.

**Quantifying Acceleration:** We illustrate in Figure 2, that indeed using mixture normalization accelerates the convergence and results in lower final test error. Table 2 also indicates the same when we report the maximum test accuracy after 25%, 50%, 75% and 100% of total number of epochs. To provide a precise comparison, we follow Ioffe and Szegedy [1] and report the number of steps (gradient descent updates) for mixture normalization variants in order to reach the maximum test accuracy achieved by their batch-normalized counterparts. Table 3 shows that on CIFAR-10, mixture normalization reduces the number of

CIFAR-10						
model	mixture norm. setting	training setting	maximum test accuracy(%) after			
	(layer, K, EM iter.)	(lr, weight decay)	25 epochs	50 epochs	75 epochs	100 epochs
BN-1	–	(0.01, 1e-4)	79.37	84.34	86.24	86.74
BN-2	–	(0.01, 2e-5)	84.11	86.92	87.65	87.95
BN-3	–	(0.05, 1e-4)	68.77	71.68	75.25	77.35
BN-4	–	(0.05, 2e-5)	73.84	77.84	81.43	82.58
MN-1	(conv3, 3, 2)	(0.01, 1e-4)	81.91	85.10	86.51	87.08
MN-2	(conv3, 3, 2)	(0.01, 2e-5)	85.56	87.28	88.05	88.47
MN-3	(conv3, 3, 2)	(0.05, 1e-4)	67.84	73.09	74.80	77.33
MN-4	(conv3, 3, 2)	(0.05, 2e-5)	76.70	80.82	83.31	83.87
CIFAR-100						
model	mixture norm. setting	training setting	maximum test accuracy(%) after			
	(layer, K, EM iter.)	(lr, weight decay)	25 epochs	50 epochs	75 epochs	100 epochs
BN-1	–	(0.01, 1e-4)	52.54	57.83	60.37	62.11
BN-2	–	(0.01, 2e-5)	56.09	60.05	61.46	62.14
BN-3	–	(0.05, 1e-4)	23.13	28.12	33.21	36.80
BN-4	–	(0.05, 2e-5)	36.98	43.89	46.95	49.08
MN-1	(conv3, 3, 2)	(0.01, 1e-4)	55.30	60.47	61.46	62.20
MN-2	(conv3, 3, 2)	(0.01, 2e-5)	59.04	60.86	61.77	62.29
MN-3	(conv3, 3, 2)	(0.05, 1e-4)	29.80	36.30	40.66	43.77
MN-4	(conv3, 3, 2)	(0.05, 2e-5)	42.31	48.96	51.98	52.62
MN-5	(conv3, 3, 4)	(0.01, 2e-5)	59.05	61.94	62.56	62.76
MN-6	(conv3, 3, 8)	(0.01, 2e-5)	58.98	61.83	62.67	63.12
MN-7	(conv2, 3, 2)	(0.01, 2e-5)	57.63	61.23	62.15	62.63
MN-8	((conv2,conv3), 3, 2)	(0.01, 2e-5)	59.79	61.67	62.50	62.96

TABLE 2: Experiments on CIFAR-10 and CIFAR-100 using CIFAR CNN architecture (ref. Table 1). We observe that irrespective of the weight decay and learning rate, not only MN models converge faster but also achieve better final test accuracy, compared to their corresponding BN counterparts. When mixture normalization is applied to multiple layers (*i.e* MN-8), we use the same K and EM iter. values for all the corresponding layers.

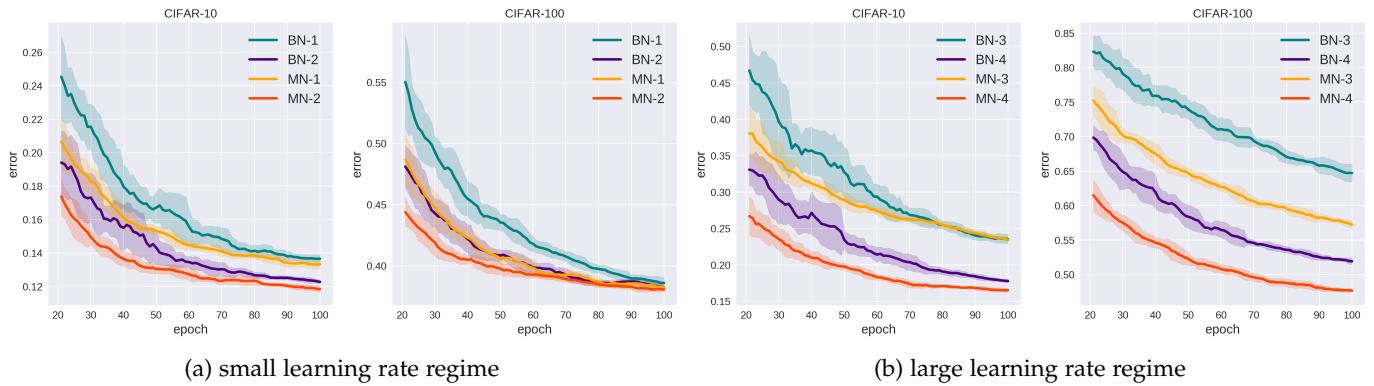


Fig. 2: Test error curves when CIFAR CNN architecture (ref. Table 1) is trained under different learning rate and weight decay settings. We observe that on CIFAR-10 and CIFAR-100, MN performs consistently in both small and large learning rate regimes.

training steps in order to reach 87.95% test accuracy by  $\sim 31\%$ . Similarly, on CIFAR-100, the best performing variant of mixture normalization, MN-6, reduces the number of training steps in order to reach 62.14% test accuracy by  $\sim 40\%$ , meanwhile its very own maximum test accuracy outperforms BN-2 by  $\sim 1\%$ . This further affirms that, our proposed mixture normalization is not only effectively accelerating the training procedure, but also reaches better local minima.

### 4.3 Inception-V3

So far, we have shown that mixture normalization can improve the convergence rate of batch-normalized models. However, our experiments were conducted on a shallow 5-layers deep architecture. Hence, it is reasonable to question whether the same behavior can be observed in very deep and more modern architectures? To address this, we choose Inception-V3 [2]. Its architecture is 48 layers deep and uses global average pooling instead of fully-connected layers, which allows operating on arbitrary input image sizes. Inception-V3 [2] has a total output stride of 32. However, to maintain low computation cost and memory utilization,

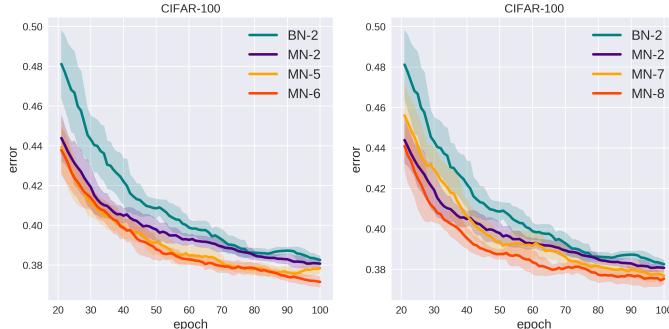


Fig. 3: Left: effect of the number of EM iterations on test error. Right: effect of utilizing MN at different layers, on test error. We show that more EM iterations and utilizing MN at multiple layers, increase the convergence rate of mixture normalized models.

CIFAR-10		
model	steps to 87.95%	max. acc. (%)
BN-2	$1.95 \times 10^4$	87.95
MN-2	$1.34 \times 10^4$	88.47
CIFAR-100		
model	steps to 62.14%	max. acc. (%)
BN-2	$1.81 \times 10^4$	62.14
MN-2	$1.77 \times 10^4$	62.29
MN-5	$1.07 \times 10^4$	62.76
MN-6	$1.09 \times 10^4$	63.12
MN-7	$1.34 \times 10^4$	62.63
MN-8	$1.03 \times 10^4$	62.96

TABLE 3: For batch normalization and the mixture-normalized variants using CIFAR CNN architecture (ref. Table 1), the number of training steps required to reach the maximum accuracy of batch-normalized model alongside with the maximum accuracy achieved by each variant.

the size of activation maps quickly reduces by a factor of 8 in only first seven layers. This is done by one convolution and two max pooling layers that operate with the stride of 2. The network is followed by three blocks of Inception separated by two grid reduction modules. Each Inception block consists of multiple Inception layers that are sequentially stacked. Specifically, first, second and third Inception blocks are respectively comprised of 3, 4 and 2 Inception layers. Spatial resolution of the activations remains intact within the Inception blocks, while grid reduction modules halve the activation size and increase the number of channels.

To make Inception-V3 [2] architecture effectively applicable to images in CIFAR-10 and CIFAR-100, that are only  $32 \times 32$ , we need to slightly modify the architecture. Specifically, we change the stride of the first convolution layer from 2 to 1 and remove the first max pooling layer. This way, the output stride of Inception-V3 architecture reduces to 8. That is, activations maintain sufficient resolution throughout the network’s depth, with the final activation (before the global average pooling) be of size  $3 \times 3$ . From now on, when we refer to the Inception-V3, we mean this modified version.

To train our models, we use RMSprop [31] with 0.9 as momentum and exponentially reduce the initial learning

rate every four epochs with the decay rate of 0.93. The size of mini-batch is set to 128, weight decay to 0.0005 and we train all the models for 200 epochs. In our preliminary experiments, we observed that learning rate of 0.001 gives the best final test accuracy for batch normalized models. Therefore, we use it for all the experiments except in one case which aims at analyzing large learning rate regime.

Table 4 and Figure 4 compare our proposed mixture normalization with its standard batch normalization counterparts. We observe that, with the exception of MN-1 on CIFAR-100, all the MN variants are not only successfully accelerating BN variants, but also achieve superior final test accuracy. As expected, and similar to the shallow network case, there is a benefit in using mixture normalization at multiple layers in different depth. Table 4 also recommends using mixture normalization in later layers, closer to the classifier, in order to reach a better convergence rate and final test accuracy.

**Effect of parameters:** We can observe from MN-3 versus MN-4 on CIFAR-10, that mixture normalization’s performance is not sensitive to the number of mixture components assuming K is large enough. We have discussed above that the number of EM iterations directly affect the computation time. However, Table 4 and Figure 4, demonstrate that by employing mixture normalization even with 2 EM updates, training a modern 48-layers deep architecture such as Inception-V3 will enjoy decent acceleration.

**Large learning rate:** To analyze the behavior of mixture normalization in large learning rate regime, we experiment training our models with the learning rate of 0.005 ( $5 \times 0.001$ ). Figure 4c illustrates that MN-5 handles large learning rate better than BN-5, its batch normalized counterpart. Note that, to probe the sole effect of increasing the learning rate, here we keep the same learning rate decay policy (decay rate of 0.93 every four epochs) as the small learning rate regime.

**Quantifying Acceleration:** Similar to the case of CIFAR CNN architecture, we report the number of steps (gradient descent updates) mixture normalization variants require in order to reach the maximum test accuracy obtained by their batch-normalized counterparts. Table 5 shows that the best performing variants of mixture normalization, MN-2 on CIFAR-10 and MN-4 on CIFAR-100, reduce the number of training steps towards 91.50% and 71.30% test accuracy, respectively on CIFAR-10 and CIFAR-100, by  $\sim 47\%$ . Similar to the case of shallow CIFAR CNN architecture, here, mixture normalization, also improves the final test accuracy by  $\sim 1.5\%$ .

**Choice of optimization technique:** We evaluate whether the performance of mixture normalization is consistent using different choices of optimization technique. That is, instead of RMSprop [31], we use Nesterov’s accelerated gradient [34], with 0.9 as momentum and decay the learning rate in two different fashions. First, following the policy adopted in ResNet [4] and DenseNet [5], we reduce the learning rate twice by a factor of 10 at 50% and 75% of the total number of epochs. Second, following Wide Residual networks [28], we reduce the learning rate three times with a factor of 5 at 30%, 60% and 80% of the total number of epochs. For MN models, mixture normalization is employed at “inc2/1” and “inc3/0” layers. Mini-batch size and the

CIFAR-10						
model	mixture norm. setting	training setting	maximum test accuracy(%) after			
	(layer, K, EM iter.)	(lr, weight decay)	50 epochs	100 epochs	150 epochs	200 epochs
BN-1	-	(0.001, 5e-4)	89.18	90.62	91.08	91.50
MN-1	((red1,red2), 3, 8)	(0.001, 5e-4)	89.47	91.03	91.70	91.91
MN-2	((red1,red2,inc3/0), 3, 4)	(0.001, 5e-4)	89.73	91.69	92.09	92.55
MN-3	((inc2/0,inc3/0), 3, 2)	(0.001, 5e-4)	90.49	91.52	92.17	92.30
MN-4	((inc2/0,inc3/0), 5, 2)	(0.001, 5e-4)	90.26	91.77	92.00	92.25

CIFAR-100						
model	mixture norm. setting	training setting	maximum test accuracy(%) after			
	(layer, K, EM iter.)	(lr, weight decay)	50 epochs	100 epochs	150 epochs	200 epochs
BN-1	-	(0.001, 5e-4)	66.88	69.44	71.03	71.30
BN-5	-	(0.005, 5e-4)	40.68	47.47	51.13	52.63
MN-1	((red1,red2), 3, 8)	(0.001, 5e-4)	66.6	68.86	70.01	70.43
MN-2	((red1,red2,inc3/0), 3, 4)	(0.001, 5e-4)	68.26	70.78	71.78	72.23
MN-4	((inc2/0,inc3/0), 5, 2)	(0.001, 5e-4)	68.4	70.99	71.98	72.74
MN-5	((inc2/0,inc3/0), 5, 2)	(0.005, 5e-4)	50.41	55.69	57.88	59.10

TABLE 4: Experiments on CIFAR-10 and CIFAR-100 using Inception-V3 architecture. Notation: “red1” (“red2”) refers to the first (second) grid reduction modules. Similarly “inc2/0” (“inc3/0”) refers to the first inception layer in second (third) inception block of the architecture. In MN-\* settings, we only replace the last batch normalization in each branch of the corresponding Inception layer with our mixture normalization. When mixture normalization is applied to multiple layers, we use the same K and EM iter. values for all the corresponding layers.

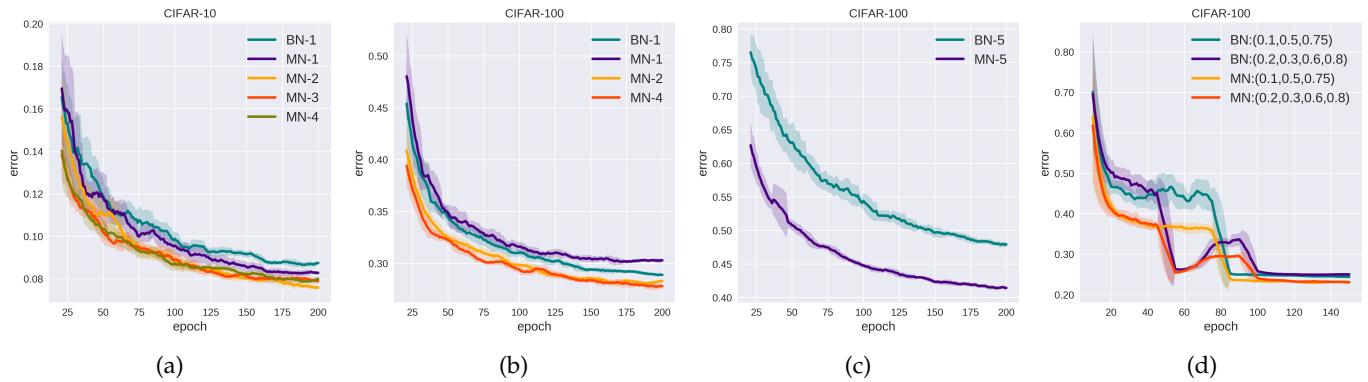


Fig. 4: Test error curves when Inception-V3 architecture is trained under different settings. Figures 4a and 4b show the small learning rate regime, respectively, on CIFAR-10 and CIFAR-100. Figure 4c shows the large learning rate regime on CIFAR-100. Figure 4d illustrates test error curves of CIFAR-100 when Inception-V3 architecture is trained using Nesterov’s accelerated gradient [34] (all other experiments use RMSprop [31]), with two different learning rate drop policies. Mixture normalization modules have been employed in “inc2/1” and “inc3/0” layers. We observe that across a variety of choices such as the number of mixture components, number of EM iterations, learning rate regime and drop policy, optimization technique, and the layer where MN is applied, mixture normalized models consistently accelerate their batch normalized counterparts and achieve better final test accuracy.

initial learning rate are respectively set to 256 and 0.14 and we train all the models for 150 epochs.

Figure 4d shows the test error curves on CIFAR-100. Similar to the previous experiments, in Table 6, we compare the required number of gradient updates in order to reach the maximum test accuracy of batch normalized model. In both learning rate decay scenarios, mixture normalization is not only able to considerably accelerate training procedure, but also achieves  $\sim 2\%$  better final test accuracy. Finally, in comparison with the state-of-the-art architectures of comparable depth, our MN model, with a test error of 22.68% on CIFAR-100, performs on par with Wide Residual networks [28] while outperforming DenseNet [5] (ref. Table 2 in [5]).

#### 4.4 DenseNet

We conclude our experimental results by evaluating the effectiveness of mixture normalization in DenseNet [5] architectures. We use two basic architectures with 40 and 100 layers. There are three dense blocks and two transition layers. Both architectures are using a growth rate of 12. All models are trained for 200 epochs on CIFAR-100, with the batch size of 64 using Nesterov’s accelerated gradient [34]. Learning rate is initialized at 0.1 and is divided by 10 at 50% and 75% of the total number of training epochs. We set the weight decay and momentum values to  $10^{-4}$  and 0.9, respectively. Figure 5 illustrates the classification error and cross entropy loss of training and test. For mixture

CIFAR-10		
model	steps to 91.50%	max. acc. (%)
BN-1	$7.34 \times 10^4$	91.50
MN-1	$4.29 \times 10^4$	91.91
MN-2	$3.86 \times 10^4$	92.55
MN-3	$3.82 \times 10^4$	92.30
MN-4	$3.58 \times 10^4$	92.25
CIFAR-100		
model	steps to 71.30%	max. acc. (%)
BN-1	$7.34 \times 10^4$	71.30
MN-1	—	70.43
MN-2	$4.57 \times 10^4$	72.23
MN-4	$3.93 \times 10^4$	72.74

TABLE 5: For batch normalization and the mixture-normalized variants using Inception-V3, the number of training steps required to reach the maximum accuracy of batch-normalized model, and the maximum accuracy achieved by each variant.

lr decay policy: (0.1,0.5,0.75)		
model	steps to 75.75%	max. acc. (%)
BN	$2.78 \times 10^4$	75.75
MN	$1.52 \times 10^4$	77.30
lr decay policy: (0.2,0.3,0.6,0.8)		
model	steps to 75.20%	max. acc. (%)
BN	$2.41 \times 10^4$	75.20
MN	$1.79 \times 10^4$	77.32

TABLE 6: Training Inception-V3 using Nesterov’s accelerated gradient [34] on CIFAR-100, the number of training steps required to reach the maximum accuracy of batch-normalized model along with the maximum accuracy achieved by each model.

normalized models, denoted by MN, we solely replace the batch normalization layers of two transition layers and the last (after the third dense block), with mixture normalization. The number of components and EM iterations for MN variants are set to 5 and 2, respectively. From Figure 5, we observe that mixture normalization not only facilitates training by accelerating the optimization, but also consistently provides better generalization on both architecture settings. In fact, the benefit of mixture normalization is more clear on the deeper setting of the DenseNet architecture. Using DenseNet {L=40, k=12} on CIFAR-100, the best test error achieved by the BN variant is 25.10% versus 24.63% of its MN counterpart. When we switch to deeper DenseNet {L=100, k=12}, architecture, this number reduces to 21.93% for BN-based model while MN variant reaches the best test error of 20.97%.

#### 4.5 Mixture Normalization in GANs

Generative adversarial networks (GANs) [27] have recently shown amazing progress in generating new realistic images [35] [36] [37] [38]. The training process is a minimax game between generator and discriminator. Discriminator learns to separate real images from the fake ones, created by the generator. Meanwhile, generator attempts to impede

discriminator’s job by progressively generating more realistic images. Hence, at convergence, generator, theoretically, must be able to generate images, whose distribution matches the distribution of the real images. However, in practice, “mode collapse” [39] [40] [41] [42] [43] [44] problem prevents generator from learning a diverse set of modes with high probability, as is in the distribution of real images. Therefore, since our proposed mixture normalization, normalizes internal activations, independently over multiple disentangled modes of variation, we hypothesize that employing it in generator, should improve the training procedure of GANs.

To evaluate our hypothesis, we consider popular Deep Convolutional GAN (DCGAN) [35] architecture. Its generator consist of one linear and four deconvolution layers. The first three deconvolution layers are separated from each other by batch normalization [1] followed by ReLU [11] activation function. We replace the batch normalization layers associated with the first two deconvolution layers with mixture normalization (K=3, EM iter=2). We train all the models on CIFAR-10 for 100K updates (iterations) using Adam [45] with  $\alpha = 0.0002$ ,  $\beta_1 = 0$  and  $\beta_2 = 0.9$  for both generator and discriminator. The quality of GANs are measured using “Fréchet Inception Distance”(FID) [46], evaluated every 10K updates for computational efficiency.

Figure 6 demonstrates that mixture normalized DCGAN (DCGAN-MN) not only converges faster than its batch normalized counterpart but also achieves better (lower) FID. While DCGAN-BN reaches the lowest FID of 37.56 (very close to 37.7 reported in [46]) after 60K steps (gradient updates). It only takes 25K steps, a reduction of  $\sim 58\%$ , for DCGAN-MN to reach 37.56. Furthermore, the lowest FID obtained by DCGAN-MN is 33.35, a significant improvement over the batch normalized model. Figure 7 illustrates samples of generated images by batch and mixture normalized DCGANs at their lowest evaluated FID.

## 5 COMPUTATIONAL COMPLEXITY AND DETAILED ANALYSIS

In this section, we give an in-depth analysis on the behavior of the mixture normalization. Specifically, we provide computational complexity analysis, both in theory and practice. We discuss how representation of mixture components evolves and whether the rationale to utilize mixture model remains valid as we pass early stages of the training process.

### 5.1 Computational Complexity Analysis

The computational overhead of mixture normalization, in comparison to the batch normalization [1], is in estimating the parameters of the Gaussian mixture model. This is a two-stage process where, we use a seeding procedure to initialize the centers of the mixture components, followed by estimating the parameters of the mixture model through iterations of Expectation-Maximization (EM).

In our implementation<sup>7</sup>, we use K-means++ [23] as the seeding procedure. Without any assumption on the data,

7. Mixture Normalization is implemented in Chainer [47], and thanks to CUPY [48], the entire computation including K-means++ and EM iterations are performed in GPU.

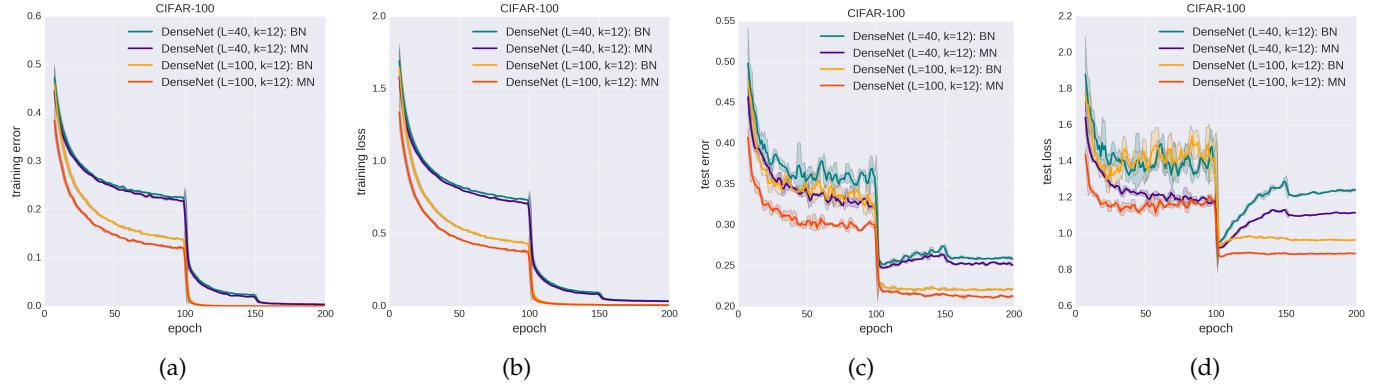


Fig. 5: DenseNet [5] experiments on CIFAR-100. Figures 5a and 5b respectively illustrate the training error and cross entropy loss. Figures 5c and 5d respectively illustrate the test error and cross entropy loss. We observe from 5a and 5b that mixture normalization facilitates the training process by accelerating the optimization. Meanwhile it provides better generalization (ref. 5c, and 5d) by continuously maintaining a large gap with respect to its batch normalized counterpart. We show 1 standard deviation (shaded area) computed within a window of 3 epochs for all the curves.

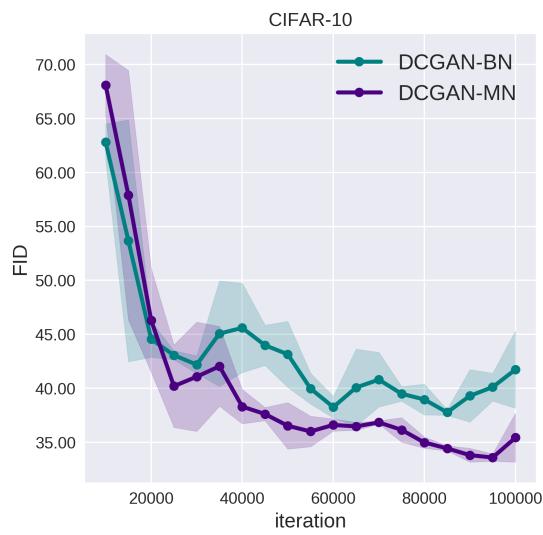
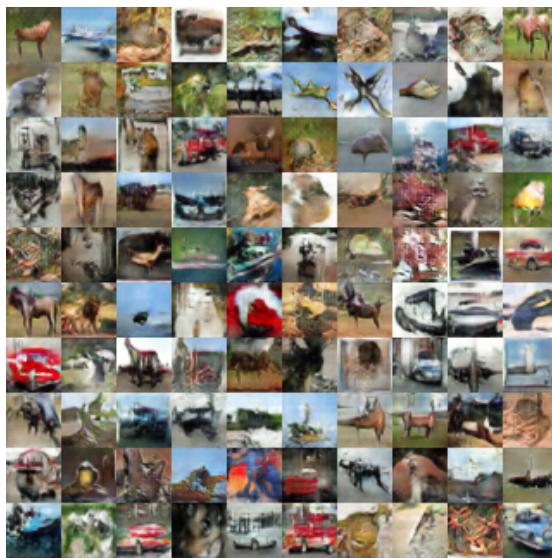


Fig. 6: Mixture normalization in deep convolutional GAN (DCGAN) [35]. We observe that employing our proposed mixture normalization in the generator of DCGAN (DCGAN-MN) facilitates the training process. In comparison with the standard DCGAN which uses batch normalization (DCGAN-BN), DCGAN-MN not only converges faster (a reduction of  $\sim 58\%$ ) but also achieves better (lower) FID (33.35 versus 37.56). For better visualization, we show one standard deviation (shaded area) computed within a window of 30K iterations (3 adjacent FID evaluation points).

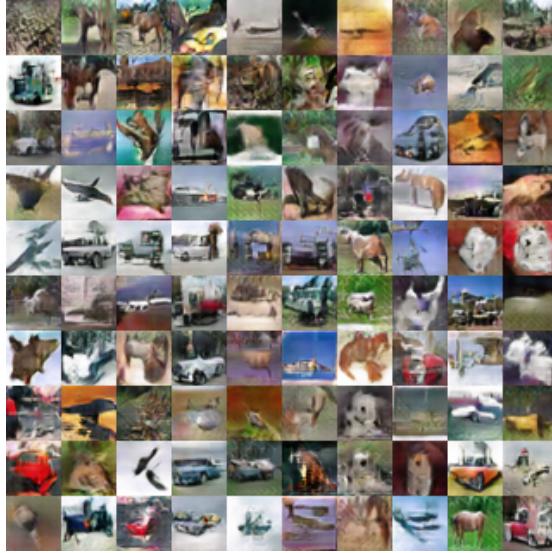
it is in expectation  $O(\log k)$ -competitive with the optimal K-means clustering solution [23], and has a complexity of  $\Theta(nkd)$  where,  $n$  is the number of data points,  $k$  is the number of cluster centers and  $d$  is the dimensionality of the data [49]. However, there is a rich literature on algorithms that speed up K-means++ [23] seeding procedure. For example, Bachem *et. al* [49] have proposed a MCMC-based sampler to approximate the full seeding step of K-means++ with complexity of  $\Theta(mk^2d)$ . Under some light assumptions, the authors prove that their proposed solution is in expectation

$O(\log k)$ -competitive with the optimal solution, if we have the chain length  $m \in \Theta(k \log^2 n \log k)$ . In this case, the total computational complexity will be  $O(k^3 dn \log k)$ , which is sublinear in the number of data points. Authors in [50] show that an assumption-free version of [49] with lower complexity is also achievable. Finally, to ameliorate the inherent sequential passes over data, Scalable K-means++ [51] with nice theoretical support [52] can be used. These all indicate that the initial seeding procedure used in mixture normalization can be implemented with low computational complexity while maintaining provably good seeding. It is easy to show that assuming diagonal covariance, each EM iteration to fit the GMM parameters is of complexity  $O(nkd)$  [53], where  $n$  can significantly be reduced using coresets [54]. In practice, we found that one does not need to use all the data points to estimate the parameters of the mixture model. Instead a simple random sampling that maintains at least 25% of the data points is sufficiently accurate. Hence, we confirm that estimating the parameters of the Gaussian mixture model can be done efficiently in scale.

In order to see how the above complexity analysis translates in practice, we compared the computation cost of mixture normalization against native implementation of batch normalization [1]. Experiments are conducted on CIFAR-100 using Densenet [5] architecture where, MN replaces the last (after the third dense block) BN layer. We tried Densenet (3 blocks and growth rate of 12) with 20 and 40 layers which result in the number of input channels to batch/mixture normalization to be respectively 196 and 448. This allows us to evaluate the effect of dimensionality of data points. We also used batch sizes of 64 and 128 in order to study the effect of the number of data points. Finally, we varied the number of mixture components ( $K$ ) from 2 to 5 to analyze its effect on the computation cost. Note that the iteration/sec. counts for the computation of the entire network not just the batch/mixture normalization layer. Table 7 indicates that in practice, the computation cost of MN scales very well with respect to the number of datapoints, dimensionality of data and the number of mixture components.



(a) DCGAN-BN



(b) DCGAN-MN

Fig. 7: Samples of generated images by batch and mixture normalized DCGAN models, at their best (lowest) evaluated FID, are respectively illustrated in Figure 7a and 7b. DCGAN-BN and DCGAN-MN, respectively achieve FID of 37.56 and 33.35.

## 5.2 Evolution of Mixture Components

To better understand how mixture normalized models evolve, we visualize in Figure 8, the mixture components associated to “inc2/0/2/4”<sup>8</sup> layer as training MN-4 (ref. Table 4) on CIFAR-100 progresses. We show a random subset of 192 channels at 20%, 40%, 60%, 80% and 100% of total training iterations.

There are two main observations here. First, in early stages of training, multiple modes captured by different mixture components are of relatively similar weights ( $\lambda_k$  in Equation 26). That is aligned with our argument that due

8. It refers to when the fifth batch normalization in the third branch of the first inception layer in second inception block is replaced with mixture normalization.

model	K	batch size	dim.	iteration/sec.
BN	1	64	196	16.40
MN	2	64	196	14.46
MN	3	64	196	14.42
MN	4	64	196	13.91
MN	5	64	196	13.69
BN	1	64	448	9.26
MN	2	64	448	8.83
MN	3	64	448	8.51
MN	4	64	448	8.52
MN	5	64	448	8.41
BN	1	128	448	7.35
MN	2	128	448	6.59
MN	3	128	448	6.51
MN	4	128	448	6.41
MN	5	128	448	6.38

TABLE 7: Computation cost comparison of mixture normalization against natively implemented (no CUDA-kernel) batch normalization [1]. Experiments are conducted on a Titan X (Pascal) GPU.

to non-linearities, underlying distributions are comprised of multiple modes of variation. However, as training procedure goes on, mixture components evolve as some get closer, while others are pushed away from each other creating more distinct components. Second, notice how the horizontal axis, associated with the activation values, reduces in range. This alongside with distribution of  $\lambda_k$ ’s morphing from relatively uniform into one with a dominant bin (component in olive), demonstrates that mixture normalization, as intended, tries to transform the underlying distribution of activations from a wide distribution comprised of multiple large components into a narrow one with a single dominant mode. Notice that despite a dominant component emerging, other components do not necessarily vanish rather their contribution diminishes. We will later show that such imbalance between  $\lambda_k$  of various components is not large enough to trigger major mode collapse. These observations confirm that our hypothesis regarding the nature of the underlying distribution is valid and mixture normalization in practice, follows what its formulation is advocating.

## 5.3 Effective Number of Mixture Components

One of the hyperparameters of mixture normalization is the number of its components, K, which must be specified as a choice of design. In some cases, K-means clustering or GMM may generate components with very small  $\lambda_k$ , meaning that the corresponding component is not very representative. Therefore, we, in our implementation, have opted heuristics to discard components whose normalized  $\lambda_k$  is less than 0.01, where samples associated to those are then merged with the remaining components. If the underlying data distribution is sufficiently well approximated using a single Gaussian distribution, which is against this paper’s proposal, then we should expect mixture normalization to not utilize all the K components. Figure 9 illustrates the actual number of components that mixture normalization has used through the entire training procedure of MN-4 (ref. Table 4) on CIFAR-10 and CIFAR-100. We show the curves associated to all the 10 mixture normalization modules in “inc2/0” and “inc3/0”. We observe that except two cases,

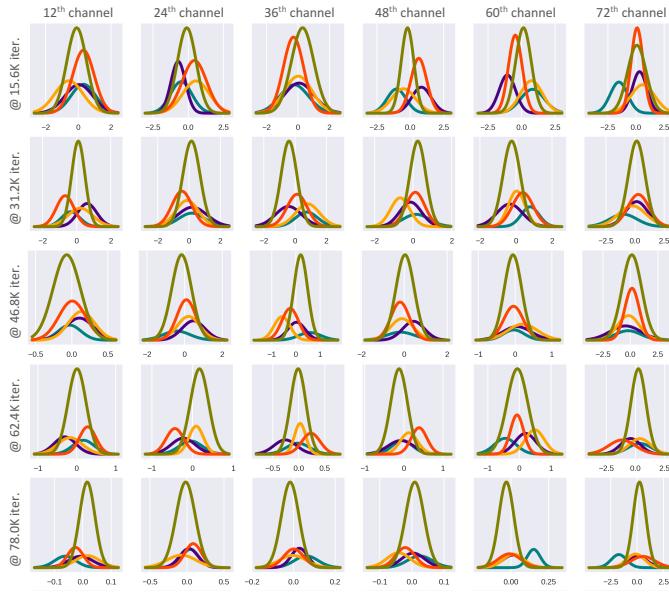


Fig. 8: Evolution of mixture normalization associated to “inc2/0/2/4” layer as training MN-4 on CIFAR-100 progresses. As argued before, we observe that the underlying distribution is comprised of multiple modes of variation. While these sub-populations are of relatively uniform importance at the beginning, as training procedure goes on, mixture components evolve where some get closer, while others are pushed away from each other creating more distinct components. Here, different colors index mixture components, when sorted according to  $\lambda_k$  values.

the rest of mixture normalization modules consistently utilize all the  $K=5$  components. This suggests that despite the potential appearance of a dominant component (ref. Section 5.2), properly estimating the underlying distribution of activations, still prefers a mixture model over a single Gaussian distribution.

## 6 CONCLUSION

Batch normalization (BN) [1] uses mean and standard deviation, computed over entire instances within a mini-batch, in order to normalize input activations to the layers. In this paper, we proposed improving BN by normalizing with respect to multiple mean and standard deviations associated with different modes of variation in the underlying data distribution. Specifically, each instance in the mini-batch, is softly assigned to a component of the Gaussian mixture model, where the GMM approximates the probability density function of input activations. Our proposed mixture normalization (MN) technique by piecewise re-structuring the probability density function of mini-batch distribution is able to drastically accelerate training deep CNN architectures. Experiments on CIFAR-10 and CIFAR-100 [26], in a variety of training scenarios, using both shallow and very deep architectures, confirm the effectiveness of mixture normalization. We have provided detailed analysis on the evolution of mixture components as they go through training procedure. Beyond image classification, inspired by the “mode collapse” problem in training GANs [27], we have

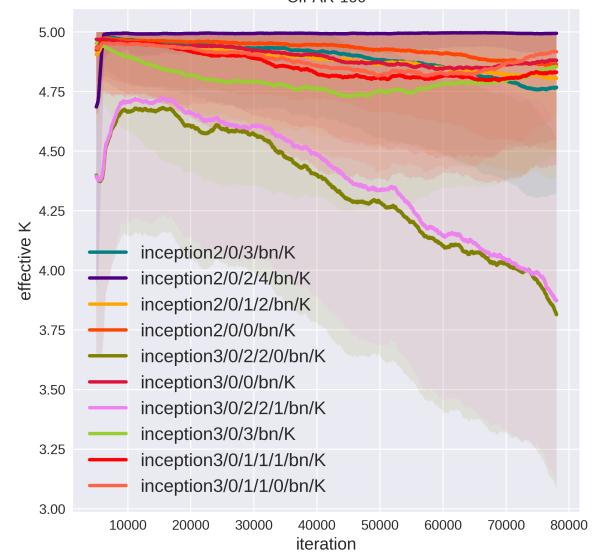
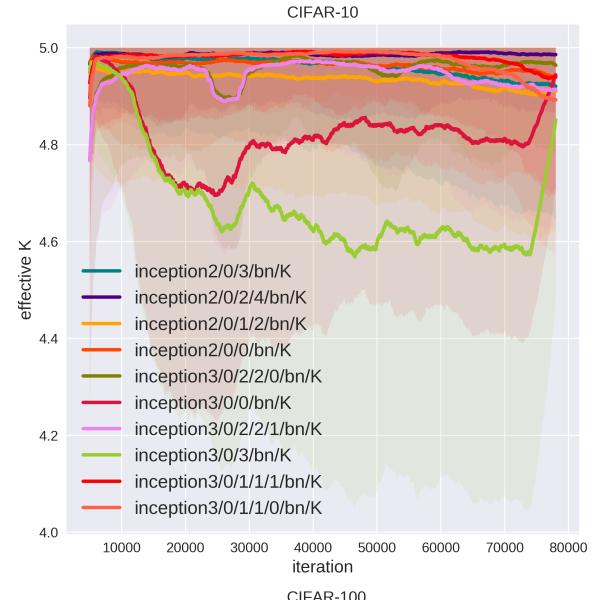


Fig. 9: Effect of the number of mixture components in MN-4 using Inception-V3 when trained on CIFAR-10 and CIFAR-100. For the sake of better visualization, curves are smoothed using running average and one standard deviation is shown as the shaded area. We can see that the majority of MN modules fully utilize all ( $K=5$ ) their mixture components, indicating that the need for better approximation using mixture model does not disappear, rather slightly diminishes, as the training procedure continues.

demonstrated that mixture normalization when integrated in the layers of generator, not only accelerates the training process of DCGAN [35] but also results in higher quality models. Hence, we affirm that in the context of deep convolutional neural networks, underlying data distribution is indeed better approximated via GMM and training benefits from such mixture modeling that MN provides. In future, we are going to study the application of mixture normalization, to the recurrent neural networks. Furthermore, we will explore estimating Gaussian mixture model, via batch and stochastic Riemannian optimization [25]. This would replace

the EM optimization, that is currently performed outside the computational graph of the deep neural networks. It also allows us to utilize full covariance matrix to better approximate distribution of the internal activations.

## ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1741431. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

## REFERENCES

- [1] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015. [1](#), [2](#), [3](#), [5](#), [7](#), [8](#), [9](#), [13](#), [14](#), [15](#), [16](#)
- [2] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826. [1](#), [2](#), [8](#), [10](#), [11](#)
- [3] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," 2017. [1](#)
- [4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. [1](#), [9](#), [11](#)
- [5] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, vol. 1, no. 2, 2017, p. 3. [1](#), [2](#), [8](#), [9](#), [11](#), [12](#), [14](#)
- [6] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016. [1](#), [3](#), [4](#)
- [7] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis." [1](#), [3](#), [4](#)
- [8] Y. Wu and K. He, "Group normalization," *arXiv preprint arXiv:1803.08494*, 2018. [1](#), [3](#), [4](#), [8](#)
- [9] M. Ren, R. Liao, R. Urtasun, F. H. Sinz, and R. S. Zemel, "Normalizing the normalizers: Comparing and extending network normalization schemes," *arXiv preprint arXiv:1611.04520*, 2016. [1](#), [3](#), [4](#), [8](#)
- [10] T. Jaakkola and D. Haussler, "Exploiting generative models in discriminative classifiers," in *Advances in neural information processing systems*, 1999, pp. 487–493. [1](#), [2](#), [4](#), [5](#)
- [11] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814. [2](#), [5](#), [7](#), [13](#)
- [12] D. M. Titterington, A. F. Smith, and U. E. Makov, *Statistical analysis of finite mixture distributions*. Wiley, 1985. [2](#), [6](#)
- [13] S. Ioffe, "Batch renormalization: Towards reducing minibatch dependence in batch-normalized models," in *Advances in Neural Information Processing Systems*, 2017, pp. 1942–1950. [2](#), [3](#), [8](#)
- [14] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint arXiv:1609.04836*, 2016. [3](#)
- [15] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: closing the generalization gap in large batch training of neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 1731–1741. [3](#)
- [16] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 901–909. [3](#)
- [17] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*. IEEE, 2016, pp. 2414–2423. [4](#)
- [18] S.-i. Amari and H. Nagaoka, *Methods of information geometry*. American Mathematical Soc., 2007, vol. 191. [4](#)
- [19] S.-I. Amari, "Natural gradient works efficiently in learning," *Neural computation*, vol. 10, no. 2, pp. 251–276, 1998. [4](#)
- [20] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek, "Image classification with the fisher vector: Theory and practice," *International journal of computer vision*, vol. 105, no. 3, pp. 222–245, 2013. [5](#), [6](#)
- [21] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the em algorithm," *Neural computation*, vol. 6, no. 2, pp. 181–214, 1994. [6](#)
- [22] F. Perronnin and C. Dance, "Fisher kernels on visual vocabularies for image categorization," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE, 2007, pp. 1–8. [6](#)
- [23] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035. [7](#), [13](#), [14](#)
- [24] 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)*, pp. 1–38, 1977. [7](#)
- [25] R. Hosseini and S. Sra, "An alternative to em for gaussian mixture models: Batch and stochastic riemannian optimization," *arXiv preprint arXiv:1706.03267*, 2017. [7](#), [16](#)
- [26] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009. [8](#), [9](#), [16](#)
- [27] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680. [9](#), [13](#), [16](#)
- [28] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016. [9](#), [11](#), [12](#)
- [29] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *European Conference on Computer Vision*. Springer, 2016, pp. 646–661. [9](#)
- [30] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013. [9](#)
- [31] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012. [9](#), [11](#), [12](#)
- [32] M. Hardt, B. Recht, and Y. Singer, "Train faster, generalize better: Stability of stochastic gradient descent," in *International Conference on Machine Learning*, 2016, pp. 1225–1234. [9](#)
- [33] S. Arora, R. Ge, B. Neyshabur, and Y. Zhang, "Stronger generalization bounds for deep nets via a compression approach," *arXiv preprint arXiv:1802.05296*, 2018. [9](#)
- [34] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 8624–8628. [11](#), [12](#), [13](#)
- [35] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015. [13](#), [14](#), [16](#)
- [36] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," *arXiv preprint*, 2016. [13](#)
- [37] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1125–1134. [13](#)
- [38] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017. [13](#)
- [39] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," *arXiv preprint arXiv:1701.04862*, 2017. [13](#)
- [40] T. Che, Y. Li, A. P. Jacob, Y. Bengio, and W. Li, "Mode regularized generative adversarial networks," *arXiv preprint arXiv:1612.02136*, 2016. [13](#)
- [41] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2016, pp. 2172–2180. [13](#)
- [42] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled generative adversarial networks," *arXiv preprint arXiv:1611.02163*, 2016. [13](#)
- [43] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems*, 2016, pp. 2234–2242. [13](#)

- [44] A. Ghosh, V. Kulharia, V. Namboodiri, P. H. Torr, and P. K. Dokania, "Multi-agent diverse generative adversarial networks," *arXiv preprint arXiv:1704.02906*, 2017. [13](#)
- [45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Proceedings of the 3rd International Conference on Learning Representations (ICLR)* *arXiv:1412.6980*, 2015. [13](#)
- [46] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in Neural Information Processing Systems*, 2017, pp. 6629–6640. [13](#)
- [47] S. Tokui, K. Oono, S. Hido, and J. Clayton, "Chainer: a next-generation open source framework for deep learning," in *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*, 2015. [Online]. Available: [http://learningsys.org/papers/LearningSys\\_2015\\_paper\\_33.pdf](http://learningsys.org/papers/LearningSys_2015_paper_33.pdf) [13](#)
- [48] R. Okuta, Y. Unno, D. Nishino, S. Hido, and C. Loomis, "Cupy: A numpy-compatible library for nvidia gpu calculations," in *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, 2017. [Online]. Available: [http://learningsys.org/nips17/assets/papers/paper\\_16.pdf](http://learningsys.org/nips17/assets/papers/paper_16.pdf) [13](#)
- [49] O. Bachem, M. Lucic, S. H. Hassani, and A. Krause, "Approximate k-means++ in sublinear time." 2016. [14](#)
- [50] O. Bachem, M. Lucic, H. Hassani, and A. Krause, "Fast and provably good seedings for k-means," in *Advances in Neural Information Processing Systems*, 2016, pp. 55–63. [14](#)
- [51] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *Proceedings of the VLDB Endowment*, vol. 5, no. 7, pp. 622–633, 2012. [14](#)
- [52] O. Bachem, M. Lucic, and A. Krause, "Distributed and provably good seedings for k-means in constant rounds," in *International Conference on Machine Learning*, 2017, pp. 292–300. [14](#)
- [53] J. J. Verbeek, N. Vlassis, and B. Kröse, "Efficient greedy learning of gaussian mixture models," *Neural computation*, vol. 15, no. 2, pp. 469–485, 2003. [14](#)
- [54] M. Lucic, M. Faulkner, A. Krause, and D. Feldman, "Training gaussian mixture models at scale via coresets," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 5885–5909, 2017. [14](#)



**Mubarak Shah** Mubarak Shah, the Trustee chair professor of computer science, is the founding director of the Center for Research in Computer Vision at the University of Central Florida (UCF). He is an editor of an international book series on video computing, was editor-in-chief of *Machine Vision and Applications* journal, and an associate editor of *ACM Computing Surveys* journal. He was the program cochair of CVPR 2008, an associate editor of the IEEE T-PAMI, and a guest editor of the special issue of the International

Journal of Computer Vision on Video Computing. His research interests include video surveillance, visual tracking, human activity recognition, visual analysis of crowded scenes, video registration, UAV video analysis, and so on. He is an ACM distinguished speaker. He was an IEEE distinguished visitor speaker for 1997-2000 and received the IEEE Outstanding Engineering Educator Award in 1997. In 2006, he was awarded a Pegasus Professor Award, the highest award at UCF. He received the Harris Corporations Engineering Achievement Award in 1999, TOKTEN awards from UNDP in 1995, 1997, and 2000, Teaching Incentive Program Award in 1995 and 2003, Research Incentive Award in 2003 and 2009, Millionaires Club Awards in 2005 and 2006, University Distinguished Researcher Award in 2007, Honorable mention for the ICCV 2005 Where Am I? Challenge Problem, and was nominated for the Best Paper Award at the ACM Multimedia Conference in 2005. He is a fellow of the IEEE, AAAS, IAPR, and SPIE.



**Mahdi M. Kalayeh** received his B.Sc. from Tehran Polytechnic (Amirkabir University of Technology) in 2009 and M.Sc. from Illinois Institute of Technology (IIT) in 2010, both in Electrical Engineering. At IIT, he was a member of Medical Imaging Research Center (MIRC) where he worked under the supervision of Dr. Jovan G. Brankov on the Generalization Evaluation of Machine Learning Numerical Observers in Image Quality Assessment. From February 2011 to June 2012, Mahdi worked as a visiting re-

searcher at Multimedia Signal Processing Research Laboratory at Electrical Engineering Department of Tehran Polytechnic under the supervision of Dr. Hamid Sheikhzadeh. Since August 2012, Mahdi is a member of Center for Research in Computer Vision (CRCV) at the University of Central Florida where he is currently a Ph.D. candidate in Computer Science and is advised by Dr. Mubarak Shah. His research is on the intersection of Computer Vision and Machine Learning, specifically, it includes Deep Learning, Visual Attribute Prediction, Semantic Segmentation, Complex Event and Action Recognition, Object Recognition and Scene Understanding. Mahdi has published several papers in conferences and journals such as CVPR, ACM Multimedia, IEEE Transactions on Cybernetics, and IEEE Transactions on Medical Imaging. He has also served as a reviewer for peer-reviewed conferences and journals including CVPR, IJCV, IEEE Transactions on Image Processing, and IEEE Transactions on Multimedia.