MODE NORMALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Normalization methods are a central building block in the deep learning toolbox. They accelerate and stabilize training, while decreasing the dependence on manually tuned learning rate schedules. When learning from multi-modal distributions, the effectiveness of batch normalization (BN), arguably the most prominent normalization method, is reduced. As a remedy, we propose a more flexible approach: by extending the normalization to more than a single mean and variance, we detect modes of data on-the-fly, jointly normalizing samples that share common features. We demonstrate that our method outperforms BN and other widely used normalization techniques in several experiments, including single and multi-task datasets.

1 INTRODUCTION

A fundamental challenge in optimizing deep learning models is the continuous change in input distributions at each layer, complicating the training process. Normalization methods, such as batch normalization (BN) (Ioffe & Szegedy, 2015) are aimed at overcoming this issue — often referred to as internal covariate shift (Shimodaira, 2000).[1] When applied successfully in practice, BN enables the training of very deep networks, shortens training times by supporting larger learning rates, and reduces sensitivity to parameter initializations. As a result, BN has become an integral element of many state-of-the-art machine learning techniques (He et al., 2016; Silver et al., 2017).

Despite its great success, BN has drawbacks due to its strong reliance on the mini-batch statistics. While the stochastic uncertainty of the batch statistics acts as a regularizer that can boost the robustness and generalization of the network, it also has significant disadvantages when the estimates of the mean and variance become less accurate. In particular, heterogeneous data (Bilen & Vedaldi, 2017) and small batch sizes (Ioffe, 2017; Wu & He, 2018) are reported to cause inaccurate estimations and thus have a detrimental effect on models that incorporate BN. For the former, Bilen & Vedaldi (2017) showed that when training a deep neural network on images that come from a diverse set of visual domains, each with significantly different statistics, then BN is not effective at normalizing the activations with a single mean and variance.

In this paper we relax the assumption that the entire mini-batch should be normalized with the same mean and variance. We propose a novel normalization method, mode normalization (MN), that first assigns samples in a mini-batch to different modes via a gating network, and then normalizes each sample with estimators for its corresponding mode (see Figure 1). We further show that MN can be incorporated into other normalization techniques such as group normalization (GN) (Wu & He, 2018) by learning which filters should be grouped together. The proposed methods can easily be implemented as layers in standard deep learning libraries, and their parameters are learned jointly with the other parameters of the network in an end-to-end manner. We evaluate MN on multiple classification tasks and demonstrate that it achieves a consistent improvement over BN and GN.

In Section 2, we present how this paper is related to previous work. We then review BN and GN, and introduce our method in Section 3. The proposed methods are evaluated on multiple benchmarks in Section 4, and our findings are summarized in Section 5.

[1] Note that the underlying mechanisms are still being explored from a theoretical perspective, see Kohler et al. (2018); Santurkar et al. (2018).

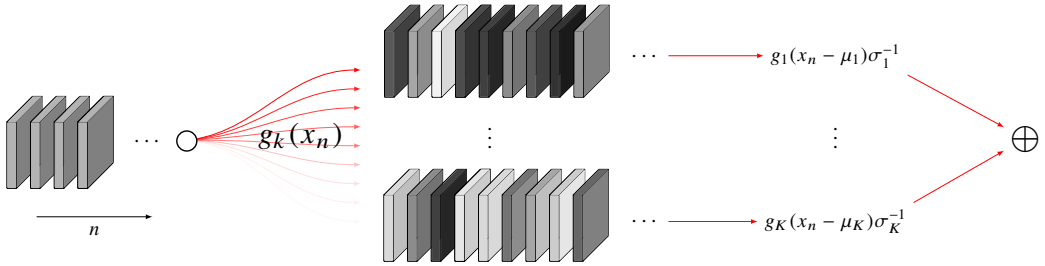

Figure 1: In mode normalization, incoming samples $\{x_n\}_{n=1,\ldots,N}$ are weighted by a set of gating functions $\{g_k\}_{k=1,\ldots,K}$. Gated samples contribute to component-wise estimators μ_k and σ_k, under which the data is normalized. After a weighted summation, the batch is passed on to the next layer. Note that during inference, estimators are computed from running averages instead.

2 Related work

Normalization. Normalizing input data (LeCun et al., 1998) or initial weights of neural networks (Glorot & Bengio, 2010) are known techniques to support faster model convergence, and were studied extensively in previous work. More recently, normalization has been evolved into functional layers to adjust the internal activations of neural networks. Local response normalization (LRN) (Lyu & Simoncelli, 2008; Jarrett et al., 2009) is used in various models (Krizhevsky et al., 2012; Sermanet et al., 2014) to perform normalization in a local neighborhood, and thereby enforce competition between adjacent pixels in a feature map. BN (Ioffe & Szegedy, 2015) implements a more global normalization along the batch dimension. In contrast to LRN, BN requires two distinct train and inference modes. At training time, samples in each batch are normalized with the batch statistics, while during inference samples are normalized using precomputed statistics from the training set. Small batch sizes or heterogeneity can lead to inconsistencies between training and test data. Our proposed method alleviates such issues by better dealing with different modes in the data, simultaneously discovering these and normalizing the data accordingly.

Several recent normalization methods (Ba et al., 2016; Ulyanov et al., 2017; Ioffe, 2017) have emerged that perform normalization along the channel dimension (Ba et al., 2016), or over a single sample (Ulyanov et al., 2017) to overcome the limitations of BN. Ioffe (2017) proposes a batch renormalization strategy that clips gradients for estimators by using a predefined range to prevent degenerate cases. While these methods are effective for training sequential and generative models respectively, they have not been able to reach the same level of performance as BN in supervised classification. Simultaneously to these developments, BN has started to attract attention from theoretical viewpoints (Kohler et al., 2018; Santurkar et al., 2018).

More recently, Wu and He (Wu & He, 2018) have proposed a simple yet effective alternative to BN by first dividing the channels into groups and then performing normalization within each group. The authors show that group normalization (GN) can be coupled with small batch sizes without any significant performance loss, and delivers comparable results to BN when the batch size is large. We build on this method in Section 3.2, and show that it is possible to automatically infer filter groupings. An alternative normalization strategy is to design a data independent reparametrization of the weights in a neural network by implicitly whitening the representation obtained at each layer (Desjardins et al., 2015; Arpit et al., 2016). While these methods show promising results, they do not generalize to arbitrary non-linearities and layers.

Mixtures of experts. Mixtures of experts (MoE) (Jacobs et al., 1991; Jordan & Jacobs, 1994) are a family of models that involve combining a collection of simple learners to split up the learning problem. Samples are thereby allocated to differing subregions of the model that are best suited to deal with a given example. There is a vast body of literature describing how to incorporate MoE with different types of expert architectures such as SVMs (Collobert et al., 2002), Gaussian processes (Tresp, 2001), or deep neural networks (Eigen et al., 2013; Shazeer et al., 2017). Most

similar to ours, Eigen et al. (2013) propose to use a different gating network at each layer in a multilayer network to enable an exponential number of combinations of expert opinions. While our method also uses a gating function at every layer to assign the samples in a mini-batch to separate modes, it differs from the above MoE approaches in two key aspects: (i.) we use the assignments from the gating functions to normalize the data within a corresponding mode, (ii.) the normalized data is forwarded to a common module (*i.e.* a convolutional layer) rather than to multiple separate experts.

Our method is also loosely related to Squeeze-and-Excitation Networks (Hu et al., 2018), that adaptively recalibrate channel-wise feature responses with a gating function. Different to their approach, we use the outputs of the gating function to normalize the responses within each mode.

Multi-domain learning. Our approach also relates to methods that parametrize neural networks with domain-agnostic and specific layers, and transfer the agnostic parameters to the analysis of very different types of images (Bilen & Vedaldi, 2017; Rebuffi et al., 2017; 2018). In contrast to these methods, which require the supervision of domain knowledge to train domain-agnostic parameters, our method can automatically learn to discover modes both in single and multi-domain settings, without any supervision.

3 METHOD

We first review the formulations of BN and GN in Section 3.1, and introduce our method in Section 3.2.

3.1 BATCH AND GROUP NORMALIZATION

Our goal is to learn a prediction rule $f\colon \mathcal{X} \to \mathcal{Y}$ that infers a class label $y \in \mathcal{Y}$ for a previously unseen sample $x \in \mathcal{X}$. For this purpose, we optimize the parameters of f on a training set $\{x_i\}_{i=1,\dots,N_d}$ for which the corresponding label information $\{y_i\}_{i=1,\dots,N_d}$ is available, where N_d denotes the number of samples in the data.

Without loss of generality, in this paper we consider image data as the input, and deep convolutional neural networks as our model. In a slight abuse of notation, we also use the symbol x to represent the features computed by layers within the deep network, producing a three-dimensional tensor $\mathcal{X} = \mathcal{C} \times \mathcal{H} \times \mathcal{W}$ where the dimensions indicate the number of feature channels, height and width respectively. Batch normalization (BN) computes estimators for the mini-batch $\{x_n\}_{n=1,\dots,N}$ (usually $N \ll N_d$) by average pooling over all but the channel dimensions.[2] Then BN normalizes the samples in the batch as

$$\mathrm{BN}(x_n) = \alpha\Big(\frac{x_n - \mu}{\sigma}\Big) + \beta, \tag{1}$$

where μ and σ are the mean and standard deviation of the mini-batch, respectively. The parameters α and β are $|\mathcal{C}|$-dimensional vectors representing a learned affine transformation along the channel dimensions, purposed to retain each layer's representative capacity (Ioffe & Szegedy, 2015). This normalizing transformation ensures that the mini-batch has zero mean and unit variance when viewed along the channel dimensions.

Group normalization (GN) performs a similar transformation to that in (1), but normalizes along different dimensions. As such, GN first separates channels $c = 1, \dots, |\mathcal{C}|$ into fixed groups G_j, over which it then jointly computes estimators, *e.g.* for the mean $\mu_j = |G_j|^{-1} \sum_{x_c \in G_j} x_c$. Note that GN does not average the statistics along the mini-batch dimension, and is thus of particular interest when large batch sizes become a prohibitive factor.

A potential problem when using GN is that channels that are being grouped together might get prevented from developing distinct characteristics in feature space. In addition, computing estimators

[2] How estimators are computed is what differentiates many of the normalization techniques currently available. Wu & He (2018) provide a detailed introduction.

Algorithm 1 Mode normalization, training phase.

Input: parameters λ, K, batch of samples $\{x_n\}$, small ε, learnable $\alpha, \beta, \Psi\colon \mathcal{X} \to \mathbb{R}^K$.

Compute expert assignments:

$$g_{nk} \leftarrow \big[\sigma \circ \Psi(x_n)\big]_k$$

for $k = 1$ to K **do**

Determine new component-wise statistics:

$$N_k \leftarrow \textstyle\sum_n g_{nk}$$
$$\langle x \rangle_k \leftarrow \frac{1}{N_k} \textstyle\sum_n g_{nk} x_n$$
$$\langle x^2 \rangle_k \leftarrow \frac{1}{N_k} \textstyle\sum_n g_{nk} x_n^2$$

Update running means:

$$\overline{\langle x \rangle}_k \leftarrow \lambda \langle x \rangle_k + (1-\lambda) \overline{\langle x \rangle}_k$$
$$\overline{\langle x^2 \rangle}_k \leftarrow \lambda \langle x^2 \rangle_k + (1-\lambda) \overline{\langle x^2 \rangle}_k$$

end for
for $n = 1$ to N **do**

Normalize samples with component-wise estimators:

$$\mu_k \leftarrow \langle x \rangle_k$$
$$\sigma_k^2 \leftarrow \langle x^2 \rangle_k - \langle x \rangle_k^2$$
$$y_{nk} \leftarrow g_{nk} \frac{x_n - \mu_k}{\sqrt{\sigma_k^2 + \varepsilon}}$$

end for
Return: $\{\alpha \sum_k y_{nk} + \beta\}_{n=1,\ldots,N}$

Algorithm 2 Mode normalization, test phase.

Input: refer to Alg. 1.

Compute expert assignments:

$$g_{nk} \leftarrow \big[\sigma \circ \Psi(x_n)\big]_k$$

for $n = 1$ to N **do**

Normalize samples with running average of component-wise estimators:

$$\mu_k \leftarrow \overline{\langle x \rangle}_k$$
$$\sigma_k^2 \leftarrow \overline{\langle x^2 \rangle}_k - \overline{\langle x \rangle}_k^2$$
$$y_{nk} \leftarrow g_{nk} \frac{x_n - \mu_k}{\sqrt{\sigma_k^2 + \varepsilon}}$$

end for
Return: $\{\alpha \sum_k y_{nk} + \beta\}_{n=1,\ldots,N}$

Algorithm 3 Mode group normalization.

Input: parameter K, sample $x \in \mathcal{C}$, small ε, learnable $\alpha, \beta, \Psi\colon \mathbb{R} \to \mathbb{R}^K$.

Compute channel-wise gates:

$$g_{ck} \leftarrow \big[\sigma \circ \Psi(x_c)\big]_k$$

for $k = 1$ to K **do**

Update estimators and normalize:

$$\mu_k \leftarrow \langle x \rangle_k$$
$$\sigma_k^2 \leftarrow \langle x^2 \rangle_k - \langle x \rangle_k^2$$
$$y_k \leftarrow \frac{x - \mu_k}{\sqrt{\sigma_k^2 + \varepsilon}}$$

end for
Return: $\frac{\alpha}{K} \sum_k y_k + \beta$

from manually engineered rules as those found in BN and GN can be too restrictive under a number of circumstances, for example when jointly learning on multiple domains.

3.2 Mode normalization

The heterogeneous nature of complex datasets motivates us to propose a more flexible treatment of normalization. Before the actual normalization is carried out, the data is first organized into modes to which it likely belongs. To achieve this, we reformulate the normalization in the framework of mixtures of experts (MoE). In particular, we introduce a set of simple gating functions $\{g_k\}_{k=1,\ldots,K}$ where $g_k\colon \mathcal{X} \to [0,1]$ and $\sum_k g_k(x) = 1$. In mode normalization (MN, Alg. 1), each sample in the mini-batch is then normalized under voting from its gate assignment:

$$\mathrm{MN}(x_n) \triangleq \alpha \Big(\sum_{k=1}^{K} g_k(x_n) \frac{x_n - \mu_k}{\sigma_k} \Big) + \beta, \tag{2}$$

where α and β are a learned affine transformation, just as in standard BN.[3]

[3]We experimented with learning individual $\{(\alpha_k, \beta_k)\}_{k=1,\ldots,K}$ for each mode. However, we have not observed any additional gains in performance from this.

The estimators for mean μ_k and variance σ_k are computed under weighing from the gating network, *e.g.* the k'th mean is estimated from the batch as

$$\mu_k = \langle x \rangle_k = \frac{1}{N_k} \sum_n g_k(x_n) \cdot x_n, \tag{3}$$

where $N_k = \sum_n g_k(x_n)$. In our experiments, we parametrize the gating networks via an affine transformation $\Psi \colon \mathcal{X} \to \mathbb{R}^K$ which is jointly learned alongside the other parameters of the network. This transformation is followed by a softmax activation $\sigma \colon \mathbb{R}^K \to [0, 1]^K$, reminiscent of attention mechanisms (Denil et al., 2012; Vinyals et al., 2015). Note that if we set $K = 1$, or when the gates collapse $g_k(x_n) = \text{const.}\ \forall\, k, n$, then (2) becomes equivalent to BN, *c.f.* (1).

As in BN, during training we normalize samples with estimators computed from the current batch. To normalize the data during inference (Alg. 2), we keep track of component-wise running estimates, borrowing from online EM approaches (Cappé & Moulines, 2009; Liang & Klein, 2009). Running estimates are updated in each iteration with a memory parameter $\lambda \in (0, 1]$, *e.g.* for the mean:

$$\overline{\langle x \rangle}_k = \lambda \langle x \rangle_k + (1 - \lambda) \overline{\langle x \rangle}_k. \tag{4}$$

Bengio et al. (2015) and Shazeer et al. (2017) propose the use of additional losses that either prevent all samples to focus on a single gate, encourage sparsity in the gate activations, or enforce variance over gate assignments. In MN, such additional penalties are not needed. Importantly, we want MN to be able to seek out a form in which it recovers traditional BN, whenever that is the optimal thing to do. In practice, we seldom observed this behavior: gates tend to receive an even share of samples overall, and they are usually assigned to individual modes.

3.3 Mode group normalization

As discussed in Section 2, GN is less sensitive to the batch size (Wu & He, 2018). Here, we show that similarly to BN, GN can also benefit from soft assignments into different modes. In contrast to BN, GN computes averages over individual samples instead of the entire mini-batch. This makes slight modifications necessary, resulting in mode group normalization (MGN, Alg. 3). Instead of learning mappings with their preimage in $\mathcal{X}$, in MGN we learn a gating network $g \colon \mathbb{R} \to \mathbb{R}^K$ that assigns channels to modes. After average-pooling over width and height, estimators are computed by averaging over channel values $x_c \in \mathbb{R}$, for example for the mean $\mu_k = \langle x \rangle_k = C_k^{-1} \sum_c g_k(x_c) \cdot x_c$, where $C_k = \sum_c g_k(x_c)$. Each sample is subsequently transformed via

$$\text{MGN}(x) \triangleq \frac{\alpha}{K} \sum_k \frac{x - \mu_k}{\sigma_k} + \beta, \tag{5}$$

where α and β are learnable parameters for channel-wise affine transformations. One of the notable advantages of MGN (that it shares with GN) is that inputs are transformed in the same way during training and inference.

A potential risk for clustering approaches is that clusters or modes might collapse into one, as described by *e.g.* Xu et al. (2005). Although it is possible to address this with a regularizer, it has not been an issue in either MN or MGN experiments. This is likely a consequence of the large dimensionality of feature spaces that we study in this paper, as well as sufficient levels of variation in the data.

4 Experiments

We consider two experimental settings to evaluate our methods: (i.) multi-task, and (ii.) single task. All experiments use standard routines within PyTorch (Paszke et al., 2017). Source code will be made available alongside the camera ready version.

4.1 MULTI-TASK

Data. In the first experiment, we wish to enforce heterogeneity in the data distribution, *i.e.* explicitly design a distribution of the form $\mathbb{P} = \sum_d \pi_d \mathbb{P}_d$. We realize this by generating a dataset whose images come from significantly diverse distributions, combining four image datasets: (i.) **MNIST** (LeCun, 1998) which contains grayscale scans of handwritten digits. The dataset has a total of 60000 training samples, as well as 10000 samples set aside for validation. (ii.) **CIFAR-10** (Krizhevsky & Hinton, 2009) is a dataset of colored images that show real world objects of one of ten classes. It contains 50000 training and 10000 test images. (iii.) **SVHN** (Netzer et al., 2011) is a real-world dataset consisting of 73257 training samples, and 26032 samples for testing. Each image shows one of ten digits in natural scenes. (iv.) **Fashion-MNIST** (Xiao et al., 2017) consists of the same number of single-channel images as are contained in MNIST. The images contain fashion items such as sneakers, sandals, or dresses instead of digits as object classes. We assume that labels are mutually exclusive, and train a single network — LeNet (LeCun et al., 1989) with a 40-way classifier at the end — to jointly learn predictions on them.

Mode normalization. Training is carried out for 3.5 million data touches (15 epochs), with learning rate reductions by 1/10 after 2.5 and 3 million data touches, respectively. Note that training for additional epochs did not result in any notable performance gains. The batch size is set to $N = 128$, and running estimates are kept with $\lambda = 0.1$. We vary the number of modes in MN over $K = \{2, 4, 6\}$. Average performances over five random initializations as well as standard deviations are shown in Table 1. MN outperforms standard BN, as well as all other normalization methods. This shows that accounting for multiple modes is an effective way to normalize intermediate features when the data is heterogeneous.

Interestingly, increasing K does not always improve the performance. The reduction in effectiveness of higher mode numbers is likely a consequence of finite estimation, *i.e.* of computing estimates from smaller and smaller partitions of the batch, a known issue in traditional BN.[4] In all remaining trials which involve single datasets and deeper networks, we therefore fixed $K = 2$. Note that the additional overhead from coupling LeNet with MN is limited. Even in our naive implementation, setting $K = 6$ results in roughly a 5% increase in runtime.

Table 1: Test set error rates (%) of batch norm (BN), instance norm (IN) (Ulyanov et al., 2017), layer norm (LN) (Ba et al., 2016), and mode norm (MN) in the multi-task setting for a batch size of $N = 128$. Shown are average top performances over five initializations alongside standard deviations. Additional results for $N = \{256, 512\}$ are shown in the Appendix.

BN	**IN**	**LN**	**MN**	K
26.91 $\pm$ 1.08	28.87 $\pm$ 2.28	27.31 $\pm$ 0.71	<u>23.16</u> $\pm$ 1.23	2
			24.25 $\pm$ 0.71	4
			25.12 $\pm$ 1.48	6

Mode group normalization. Group normalization is designed specifically for applications in which large batch sizes become prohibitive. We therefore simulate this by reducing batch sizes to $N = \{4, 8, 16\}$, and train each model for 50 000 gradient updates. This uses the same configuration as previously, except for a smaller initial learning rate $\gamma = 0.02$, which is reduced by 1/10 after 35 000 and 42 500 updates. In GN, we allocate two groups per layer, and accordingly set $K = 2$ in MGN. As a baseline, results for BN and MN were also included. Average performances over five initializations and their standard deviations are shown in Table 2. As previously shown by Wu & He (2018), BN fails to maintain its performance when the batch size is small during training. Though MN performs slightly better than BN, its performance also degrades in this regime. GN is more robust to small batch sizes, yet MGN further improves over GN, and — by combining the advantages of GN and MN — achieves the best performance for different batch sizes among all four methods.

[4] Experiments with larger batch sizes support this argument, see Appendix.

Table 2: Test set error rates (%) for BN, MN, mode group norm (MGN) and group norm (GN) on small batch sizes. Shown are average top performances over five initializations alongside standard deviations.

N	**BN**	**MN**	**GN**	**MGN**
4	33.40 ± 0.75	32.80 ± 1.59	32.15 ± 1.10	31.30 ± 1.65
8	31.98 ± 1.53	29.05 ± 1.51	28.60 ± 1.45	26.83 ± 1.34
16	30.38 ± 0.60	28.70 ± 0.68	27.63 ± 0.45	$\underline{26.00} \pm 1.68$

4.2 SINGLE TASK

Data. Here our method is evaluated in single image classification tasks, showing that it can be used to improve performance in several recently proposed convolutional networks. For this, we incorporate MN into multiple modern architectures, first evaluating it on **CIFAR10** and **CIFAR100** datasets and later on a large-scale dataset, **ILSVRC12** (Deng et al., 2009). Differently from CIFAR10, CIFAR100 has 100 classes, but contains the same number of training images, 600 images per class. ILSVRC12 contains around 1.2 million images from 1000 object categories.

Network In Network. Since the original Network In Network (NIN) (Lin et al., 2013) does not contain any normalization layers, we modify the network architecture to add them, coupling each convolutional layer with a normalization layer (either BN or MN). We then train the resulting model on CIFAR10 and CIFAR100 for 100 epochs with SGD and momentum as optimizer, using a batch size of $N = 128$. Initial learning rates are set to $\gamma = 10^{-1}$, which we reduce by 1/10 at epochs 65 and 80 for all methods. Running averages are stored with $\lambda = 0.1$. During training we randomly flip images horizontally, and crop each image after padding it with four pixels on each side. Dropout (Srivastava et al., 2014) is known to occasionally cause issues in combination with BN (Li et al., 2018), and reducing it to 0.25 (as opposed to 0.5 in the original publication) was beneficial to performance. Note that incorporating MN with $K = 2$ into NIN adds less than 1% to the number of trainable parameters.

We report the test error rates with NIN on CIFAR10 and CIFAR100 in Table 3 (left). We first observe that NIN with BN obtains an error rate similar to that reported for the original network in Lin et al. (2013). MN ($K = 2$) achieves an additional boost of 0.4% and 0.6% over BN on CIFAR10 and CIFAR100, respectively.

Table 3: Test set error rates (%) with BN and MN for deep architectures on CIFAR10, CIFAR100. Shown are NIN (left), VGG13 (middle) and ResNet20 (right).

	Network In Network			**VGG13**		**ResNet20**		
	Lin et al.	**BN**	**MN**	**BN**	**MN**	**He et al.**	**BN**	**MN**
CIFAR10	8.81	8.82	<u>8.42</u>	8.28	<u>7.79</u>	8.75	8.44	<u>7.99</u>
CIFAR100	–	32.30	<u>31.66</u>	31.15	<u>30.06</u>	–	31.56	<u>30.53</u>

VGG Networks. Another popular class of deep convolutional neural networks are VGG networks (Simonyan & Zisserman, 2014). In particular we trained a VGG13 with BN and MN on CIFAR10 and CIFAR100. For both datasets we optimized using SGD with momentum for 100 epochs, setting the initial learning rate to $\gamma = 0.1$, and reducing it at epochs 65, 80, and 90 by a factor of 1/10. The batch size is set to $N = 128$. As before, we set the number of modes in MN to $K = 2$, and keep estimators with $\lambda = 0.1$. When incorporated into the network, MN improves the performance of VGG13 by 0.4% on CIFAR10, and gains over 1% on CIFAR100.

Residual Networks. Contrary to NIN and VGG, Residual Networks (He et al., 2016) were originally conceptualized with layer-wise batch normalizations. We trained a ResNet20 on CIFAR10 and CIFAR100 in its original architecture (*i.e.* with BN), as well as with MN ($K = 2$), see Table 3 (right). On both datasets we follow the standard training procedure and train both models for 160

epochs, with SGD as optimizer, momentum parameter of 0.9, and weight decay of 10^{-4}. Running estimates were kept with $\lambda = 0.1$, the batch size set to $N = 128$. Our implementation of ResNet20 (BN in Table 3) performs slightly better than that reported in the original publication (8.42% versus 8.82%). Replacing BN with MN achieves a notable 0.45% and 0.7% performance gain over BN in CIFAR10 and CIFAR100, respectively.

We also test our method in the large-scale image recognition task of ILSVRC12. Concretely, we replaced BN in a ResNet18 with MN ($K = 2$), and trained both resulting models on ILSVRC12 for 90 epochs. We set the initial learning rate to $\gamma = 0.1$, reducing it at epochs 30 and 60 by a factor of 1/10. SGD was used as optimizer (with momentum parameter set to 0.9, weight decay of 10^{-4}). To accelerate training we distributed the model over four GPUs, with an overall batch size of $N = 256$. As can be seen from Table 4, MN results in a small but consistent improvement over BN in terms of top-1 and top-5 errors.

Table 4: Top-1 and top-5 error rates (%) of ResNet18 on ImageNet ILSVRC12, with BN and MN.

Top-k Error	**BN**	**MN**
1	30.25	<u>30.07</u>
5	10.90	<u>10.65</u>

Qualitative analysis. In Fig. 2 we evaluated the experts $g_k(\{x_n\})$ for samples from the CIFAR10 test set in layers `conv3-64-1` and `conv-3-256-1` of VGG13, and show those samples that have been assigned the highest probability to belong to either of the $K = 2$ modes. In the normalization belonging to `conv3-64-1`, MN is sensitive to a red-blue color mode, and separates images accordingly. In deeper layers such as `conv-3-256-1`, separations seem to occur on the semantic level. In this particular example, MN separates smaller objects from such that occupy a large portion of the image.

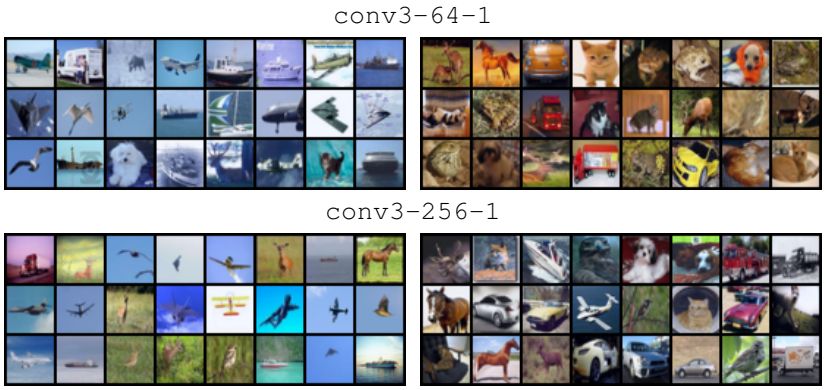

Figure 2: Test samples from CIFAR10 that were clustered together by two experts in an early layer (top) and a deeper layer (bottom) of VGG13.

5 Conclusion

Stabilizing the training process of deep neural networks is a challenging problem. Several normalization approaches that aim to tackle this issue have recently emerged, enabling training with higher learning rates, faster model convergence, and allowing for more complex network architectures.

Here, we showed that two widely used normalization techniques, BN and GN, can be extended to allow the network to jointly normalize its features within multiple modes. We further demonstrated that our method can be incorporated to various deep network architectures and improve their classification performance consistently with a negligible increase in computational overhead. As part of future work, we plan to explore customized, layer-wise mode numbers in MN, and automatically determining them, *e.g.* by utilizing concepts from sparsity regularization.

REFERENCES

Devansh Arpit, Yingbo Zhou, Bhargava Kota, and Venu Govindaraju. Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. In *Proc. ICML*, pp. 1168–1176, 2016.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

Hakan Bilen and Andrea Vedaldi. Universal representations: The missing link between faces, text, planktons, and cat breeds. *arXiv preprint arXiv:1701.07275*, 2017.

Olivier Cappé and Eric Moulines. On-line expectation–maximization algorithm for latent data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(3):593–613, 2009.

Ronan Collobert, Samy Bengio, and Yoshua Bengio. A parallel mixture of SVMs for very large scale problems. In *Proc. NIPS*, pp. 633–640, 2002.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. CVPR*, pp. 248–255. IEEE, 2009.

Misha Denil, Loris Bazzani, Hugo Larochelle, and Nando de Freitas. Learning where to attend with deep architectures for image tracking. *Neural Computation*, 24(8):2151–2184, 2012.

Guillaume Desjardins, Karen Simonyan, Razvan Pascanu, et al. Natural neural networks. In *Proc. NIPS*, pp. 2071–2079, 2015.

David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*, 2013.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proc. AISTATS*, pp. 249–256, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. CVPR*, pp. 770–778, 2016.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proc. CVPR*, 2018.

Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Proc. NIPS*, pp. 1942–1950, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. ICML*, pp. 448–456, 2015.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. *Adaptive mixtures of local experts*. MIT Press, 1991.

Kevin Jarrett, Koray Kavukcuoglu, Yann LeCun, et al. What is the best multi-stage architecture for object recognition? In *Proc. CVPR*, pp. 2146–2153. IEEE, 2009.

Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural computation*, 6(2):181–214, 1994.

Jonas Kohler, Hadi Daneshmand, Aurelien Lucchi, Ming Zhou, Klaus Neymeyr, and Thomas Hofmann. Towards a theoretical understanding of batch normalization. *arXiv preprint arXiv:1805.10694*, 2018.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS*, pp. 1097–1105, 2012.

Yann LeCun. The MNIST database of handwritten digits, http://yann.lecun.com/exdb/mnist, 1998.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

Yann LeCun, Léon Bottou, Genevieve Orr, and Klaus-Robert Müller. Efficient backprop in neural networks: Tricks of the trade. *Lecture Notes in Computer Science*, 1524, 1998.

Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. Understanding the disharmony between dropout and batch normalization by variance shift. *arXiv preprint arXiv:1801.05134*, 2018.

Percy Liang and Dan Klein. Online EM for unsupervised models. In *Proceedings of human language technologies: The 2009 annual conference of the North American chapter of the association for computational linguistics*, pp. 611–619. Association for Computational Linguistics, 2009.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Siwei Lyu and Eero P Simoncelli. Nonlinear image representation using divisive normalization. In *Proc. CVPR*, pp. 1–8. IEEE, 2008.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, 2011.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

S-A. Rebuffi, H. Bilen, and A. Vedaldi. Efficient parametrization of multi-domain deep neural networks. In *Proc. CVPR*, 2018.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *Proc. NIPS*, pp. 506–516, 2017.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? (no, it is not about internal covariate shift). *arXiv preprint arXiv:1805.11604*, 2018.

Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *Proc. ICLR*, 2014.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proc. ICLR*, 2017.

Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Volker Tresp. Mixtures of Gaussian processes. In *Proc. NIPS*, pp. 654–660, 2001.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *Proc. CVPR*, 2017.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proc. CVPR*, pp. 3156–3164. IEEE, 2015.

Yuxin Wu and Kaiming He. Group normalization. In *Proc. ECCV*, 2018.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Linli Xu, James Neufeld, Bryce Larson, and Dale Schuurmans. Maximum margin clustering. In *Proc. NIPS*, pp. 1537–1544, 2005.

A ADDITIONAL MULTI-TASK RESULTS

Shown in Table 5 are additional results for jointly training on MNIST, CIFAR10, SVHN, and Fashion-MNIST. The same network is used as in previous multi-task experiments, for hyperparameters see Section 4. In these additional experiments, we varied the batch size to $N = \{256, 512\}$. For larger batch sizes, increasing K to values larger than two increases performance, while for a smaller batch size of $N = 128$ (*c.f.* Table 1), errors incurred by finite estimation prevent this benefit from appearing.

Table 5: Test set error rates (%) of multiple normalization methods in the multi-task setting for large batch sizes. The table contains average performances over five initializations, alongside their standard deviation.

N	**BN**	**IN**	**LN**	**MN**	K
256	26.34 (1.82)	31.15 (3.46)	26.95 (2.51)	25.29 (1.31)	2
				25.04 (1.88)	4
				<u>24.88</u> (1.24)	6
512	26.51 (1.15)	29.00 (1.85)	28.98 (1.32)	26.18 (1.86)	2
				<u>24.29</u> (1.82)	4
				25.33 (1.33)	6