UAN: Unsupervised Adaptive Normalization

1st Bilal Faye
LIPN, UMR CNRS 7030
Villetaneuse, France
faye@lipn.univ-paris13.fr

2nd Hanane Azzag
LIPN, UMR CNRS 7030
Villetaneuse, France
azzag@univ-paris13.fr

3rd Mustapha Lebbah
DAVID laboratory
Versailles, France
mustapha.lebbah@uvsq.fr

4th Fangchen FANG
L2TI laboratory
Villetaneuse, France
fangchen.feng@univ-paris13.fr

Abstract—**Deep neural networks have become a staple in solving intricate problems, proving their mettle in a wide array of applications. However, their training process is often hampered by shifting activation distributions during backpropagation, resulting in unstable gradients. Batch Normalization (BN) addresses this issue by normalizing activations, which allows for the use of higher learning rates. Despite its benefits, BN is not without drawbacks, including its dependence on mini-batch size and the presumption of a uniform distribution of samples. To overcome this, several alternatives have been proposed, such as Layer Normalization, Group Normalization, and Mixture Normalization. These methods may still struggle to adapt to the dynamic distributions of neuron activations during the learning process. To bridge this gap, we introduce Unsupervised Adaptive Normalization (UAN), an innovative algorithm that seamlessly integrates clustering for normalization with deep neural network learning in a singular process. UAN executes clustering using the Gaussian mixture model, determining parameters for each identified cluster, by normalizing neuron activations. These parameters are concurrently updated as weights in the deep neural network, aligning with the specific requirements of the target task during backpropagation. This unified approach of clustering and normalization, underpinned by neuron activation normalization, fosters an adaptive data representation that is specifically tailored to the target task. This adaptive feature of UAN enhances gradient stability, resulting in faster learning and augmented neural network performance. UAN outperforms the classical methods by adapting to the target task and is effective in classification, and domain adaptation. We release our code at github repository**

I. Introduction

Deep neural networks have shown significant advancements in various domains. However, they still face specific challenges, particularly in terms of accelerating learning and improving performance. The diverse nature of data often requires pre-training methods to normalize them to the same scale and eliminate biases. In a neural network with a single layer, normalizing input data accelerates convergence, thus enhancing performance [1]. This approach, however, is limited as contemporary neural networks consist of multiple layers. Input data is connected only to the first layer of the neural network, which does not guarantee that subsequent layers will benefit from input data normalization due to weight updates during the backward pass. To address this, normalization approaches have been proposed for layers not directly connected to input data, including activation normalization, weight normalization, and gradient normalization. Batch Normalization (BN) introduced by Ioffe and Szegedy [2] normalizes the output of each neural network layer to have a zero mean and unit variance property. This approach stabilizes gradient variation, accelerating convergence and improving deep neural network performance. Despite its effectiveness, BN has limitations such as mini-batch dependency, restricting its utility for small batches, and the weak assumption that samples within the mini-batch are from the same distribution, which is not always true, especially with real-world data.

Layer Normalization (LN) [3], Instance Normalization (IN) [4], and Group Normalization (GN) [5] were proposed to address mini-batch dependency issues. Mixture Normalization (MN) [6] presents an approach based on the assumption of Gaussian mixture models (GMM) on input data to overcome the distribution assumption made by BN. MN employs a dual-stage process for normalizing neuron activations: (i)Expectation-Maximization (EM) algorithm [7] is initially used to deduce parameters for each mixture component; (ii) in the training of deep neural networks, samples linked to the same component are normalized utilizing the parameters specific to their component. This approach has demonstrated quicker convergence and enhanced performance in neural networks, particularly in tasks of supervised classification. However, MN's dependence on the EM algorithm as a preprocessing step could be a limiting factor due to its computational intensity. Additionally, MN's static parameters might not be sufficiently adaptive to the shifting distributions of neuron activations during the learning phase.

To tackle this challenge, we introduce Unsupervised Adaptive Normalization (UAN), a streamlined normalization approach. UAN is grounded in the assumption that data are governed by a Gaussian mixture model in a latent space. The objective is to discover this space during the deep neural network learning process by normalizing neuron activations. The parameters of each mixture component called ”cluster”, are estimated as weights of the deep neural network during the backpropagation phase, enabling the construction of mixture components tailored to the target task addressed by the deep neural network. This one-pass approach provides an adaptive data representation based on the clustering methodology, enhancing separability and facilitating deep neural network learning. Notably, UAN demonstrates superior results in terms of convergence and deep neural network performance when compared to BN and MN.

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

- **Simultaneous clustering for normalization and deep Learning:** Unsupervised Adaptive Normalization(UAN) integrates clustering and deep neural network learning in a single step. By normalizing neuron activations based on the Gaussian mixture model assumption, UAN estimates parameters for each mixture component, improving data representation for the target task and boosting gradient stability.
- **Efficiency and complexity reduction:** UAN stands out by simplifying the architecture and eliminating the preprocessing clustering phase using EM used in Mixture Normalization. UAN adapts normalization to the target task, determining mixture component parameters during backpropagation as model weights. This streamlined approach demonstrates superior convergence and performance compared to BN and MN.
- **Enhanced convergence and performance:** UAN surpasses Batch Normalization and Mixture Normalization regarding convergence and deep neural network performance. Its adaptive data representation, achieved through simultaneous clustering and learning, enhances separability and accelerates the learning process.

II. State of the art

To ensure clear understanding and consistency in our model, and to facilitate comparison with existing work, we adopt the same notations as those used in [6]. Let $x \in \mathbb{R}^{N \times C \times H \times W}$ neurons activations in a convolutional neural network layer, where N, C, H, and W represent the batch, channel, height, and width axes, respectively. Batch Normalization (BN) [2] standardizes a mini-batch $B = \{x_{1:m} : m \in [1, N] \times [1, H] \times [1, W]\}$ of m samples with x flattened across all axes except the channel. The standardization is expressed as:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \tag{1}$$

where $\mu_B = \frac{1}{m}\sum_{i=1}^m x_i$, $\sigma_B^2 = \frac{1}{m}\sum_{i=1}^m (x_i - \mu_B)^2$ are respectively the mean and variance. $\epsilon > 0$ is a small number to prevent numerical instability. To mitigate the constraints imposed by standardization, learnable parameters γ and β are introduced, transforming activations as follows:

$$\tilde{x}_i = \gamma \hat{x}_i + \beta. \tag{2}$$

During inference, population statistics are essential for deterministic inference. They are computed by averaging over training iterations:

$$\begin{cases} \hat{\mu} = (1 - \lambda)\hat{\mu} + \lambda \mu_B \\ \hat{\sigma}^2 = (1 - \lambda)\hat{\sigma}^2 + \lambda \sigma_B^2. \end{cases} \tag{3}$$

If the mini-batch samples come from the same distribution, the transformation in equation (1) yields a distribution with zero mean and unit variance. This constraint stabilizes activation distribution, aiding training.

The mini-batch-wise approach allows for a more suitable data representation, enabling faster processing and enhancing deep neural network performance in terms of convergence. Despite its efficacy, BN's performance is contingent on mini-batch size, and the disparity between training and inference hinders its application in intricate networks (e.g., recurrent neural networks). To address these challenges and handle parameter estimation on unrelated samples, numerous variants and alternative methods have been proposed.

Various extensions to batch normalization (BN) have emerged, including Layer Normalization (LN) [3], Instance Normalization (IN) [4], Group Normalization (GN) [5], and Mixture Normalization (MN) [6]. These alternatives share a common transformation $x \rightarrow \hat{x}$, applied to the flattened x across the spatial dimension $L = H \times W$, expressed as:

$$v_i = x_i - \mathbb{E}_{B_i}(x), \ \hat{x}_i = \frac{v_i}{\sqrt{\mathbb{E}_{B_i}(v^2) + \epsilon}}, \tag{4}$$

where $B_i = \{j : j_N \in [1, N], j_C \in [i_C], j_L \in [1, L]\}$, and $i = (i_N, i_C, i_L)$ indexing the activations $x \in \mathbb{R}^{N \times C \times L}$.

Layer Normalization (LN): Alleviating inter-dependency among batch-normalized activations, LN computes mean and variance using input from individual layer neurons. LN benefits recurrent networks but may face challenges with convolutional layers due to spatial variations in visual data. Formulated as Equation 4 with $B_i = \{j : j_N \in [i_N], j_C \in [1, C], j_L \in [1, L]\}$.

Instance Normalization (IN): This approach normalizes each sample individually, particularly aimed at eliminating image style information. IN enhances specific deep neural networks (DNNs) by computing mean values and standard deviations in the spatial domain. Applied as Equation 4 with $B_i = \{j : j_N \in [i_N], j_C \in [i_C], j_L \in [1, L]\}$.

Group Normalization (GN): GN partitions neurons into groups and autonomously standardizes layer inputs for each sample within these groups. In visual tasks with small batch sizes, GN is useful for applications like object detection and segmentation. GN calculates statistics along the L axis, restricting computation to subgroups of channels. Equivalent to IN when $G = C$ and identical to LN when $G = 1$.

Mixture Normalization (MN): In deep neural networks (DNNs), neuron activations exhibit multiple modes of variation due to non-linearities, challenging the Gaussian distribution assumption of batch normalization (BN) [2]. Mixture Normalization (MN) [6] approaches BN from a Fisher kernels perspective, using a Gaussian Mixture Model (GMM) to normalize based on multiple means and standard deviations associated with the clusters. In convolutional neural networks, MN outperforms BN in terms of convergence and accuracy in supervised learning tasks. Implemented in two stages: estimation of GMM parameters using the Expectation-Maximization (EM) algorithm [7], followed by neuron activations normalization and aggregation using posterior probabilities during deep neural network training.

The distribution p_θ characterizing the data is represented as a parameterized Gaussian Mixture Model (GMM). Let $x \in \mathbb{R}^D$, and $\theta = \{\lambda_k, \mu_k, \Sigma_k : k = 1, ..., K\}$,

$$p(x) = \sum_{k=1}^{K} \lambda_k p(x|k), \text{ s.t. } \forall_k \ : \ \lambda_k \ \geq 0, \sum_{k=1}^{K} \lambda_k = 1, \quad (5)$$

where

$$p(x|k) = \frac{1}{(2\pi)^{D/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)}{2}\right), \quad (6)$$

represents the k^{th} Gaussian in the mixture model $p(x)$, with μ_k as the mean vector and Σ_k as the covariance matrix. The posterior probability that x is generated by the k^{th} component in the mixture model is defined as:

$$\tau_k(x) = p(k|x) = \frac{\lambda_k p(x|k)}{\sum_{j=1}^{K} \lambda_j p(x|j)}, \quad (7)$$

Based on these assumptions and the general transform in Equation (4), the Mixture Normalizing Transform for a given x_i is defined as

$$\hat{x}_i = \sum_{k=1}^{K} \frac{\tau_k(x_i)}{\sqrt{\lambda_k}} \hat{x}_i^k, \quad (8)$$

given

$$\hat{x}_i^k = \frac{v_i^k}{\sqrt{\mathbb{E}_{B_i}[\hat{\tau}_k(x).(v^k)^2] + \epsilon}}, \quad (9)$$

where

$$\hat{\tau}_k(x_i) = \frac{\tau_k(x_i)}{\sum_{j \in B_i} \tau_k(x_j)},$$

and

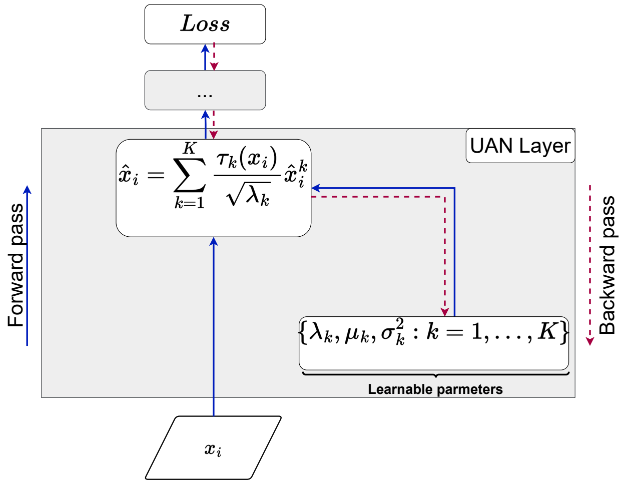
$$v_i^k = x_i - \mathbb{E}_{B_i}[\hat{\tau}_k(x).x],$$

represents the normalized contribution of x_i with respect to the mini-batch $B_i = \{j : j_N \in [i_N], j_C \in [i_C], j_L \in [1, L]\}$ in estimating the statistical measures of the k^{th} Gaussian component.

III. Contribution: Unsupervised Adaptive Normalization

Mixture Normalization (MN) effectively stabilizes deep neural network training, but it encounters limitations in its two-stage process, which involves the Expectation-Maximization (EM) algorithm, introducing complexities. The static parameters of MN do not adapt to the changing distributions of neuron activations during learning and weight updates. Recognizing the need for a more adaptive and streamlined approach, we propose Unsupervised Adaptive Normalization (UAN). In response to these challenges, UAN operates as a one-stage algorithm, embodying an ”unsupervised” approach where prior knowledge of mixture components is absent. In this context, ”clusters” refers to a mixture component within the latent space modeled as a Gaussian mixture. UAN estimates mixture component parameters as network weights, dynamically adjusting to the target task and evolving alongside deep neural network weights. This unique one-stage algorithm achieves mixture parameter estimation and target task resolution during a unified training process. Unlike MN, where normalization parameters remain constant, UAN updates these parameters throughout learning, ensuring the formation of adaptive clusters in line with the evolving distribution of neuron activations.

In implementing this approach, we assume the presence of a latent space where clusters are modeled as a mixture model. Specifically, we consider a Gaussian mixture model with parameters $\theta = \{\lambda_k, \mu_k, \Sigma_k : k = 1, ..., K\}$, where K denotes the total number of defined clusters. It is important to note that the assumption of a Gaussian distribution is not mandatory; alternative distributions can be explored. In such cases, the corresponding parameters are estimated accordingly. The normalization of each neuron activation takes place using Equation 8, where all clusters contribute to the normalization process for each activation.



$Loss$
...
UAN Layer
$\hat{x}_i = \sum_{k=1}^{K} \frac{\tau_k(x_i)}{\sqrt{\lambda_k}} \hat{x}_i^k$
Forward pass
Backward pass
$\{\lambda_k, \mu_k, \sigma_k^2 : k = 1, \ldots, K\}$
Learnable parmeters
x_i

Fig. 1: Unsupervised Adaptive Normalization (UAN) applied on neuron activation x_i. The parameter K corresponds to the number of clusters estimated during the training process. In the forward pass, the K cluster parameters $\{\lambda_k, \mu_k, \sigma_k^2\}$ are used to normalize the activation x_i through Equation 8. The resulting normalized activation serves as input for the Deep Neural Network (DNN). After loss computation, in the backward pass, the parameters $\{\lambda_k, \mu_k, \sigma_k : k = 1, ..., T\}$ undergo updates based on the target task.

Throughout the training of the deep neural network, cluster parameters are set at the beginning to fulfill the conditions $\sum_{i=1}^{K} \lambda_k = 1$ and $\sigma_k > 0$, where these parameters act as weights. To adapt these cluster parameters, we use backpropagation and a moving average technique within mini-batches (Figure 1 and Algorithm 1). The former aligns neuron activations with the target task, dynamically evolving parameters alongside deep neural network weights. Simultaneously, the latter serves as an approximation of the EM approach tailored to neuron activations. This one-stage process in UAN allows the simultaneous estimation of mixture component parameters and resolution of the target task in deep neural networks.

Unlike methods with static parameters that struggle to adapt during learning, UAN demonstrates adaptability and efficiency in overcoming evolving neuron activation distributions and weight updates.

Algorithm 1: Unsupervised Adaptive Normalization Layer

```
Input  : Deep neural network $Net$ with trainable
         parameters $\Theta$; subset of activations $\{x_i\}_{i=1}^{m}$;
         number of clusters $K$; momentum $m$;
         $update \in \{weight, moving\ average\}$
Output: Unsupervised Adaptive-Normalized deep
         neural network for inference, $Net_{UAN}^{inf}$
$Net_{UAN}^{tr} = Net$ // Training UAN deep neural network
Initialize the parameters for each cluster as follows:
 $(\lambda_k,\, \mu_k,\, \sigma_k^2)$ for $k \in \{1, ..., K\}$, subject to the
 conditions that $\sum_{k=1}^{K} \lambda_k = 1$ and $\sigma_k > 0$ for all
 $k \in \{1, ..., K\}$.
for $i \leftarrow 1$ to $m$ do
    • Add transformation $\hat{x}_i = UAN(x_i)$ to $Net_{UAN}^{tr}$ using
      Equation 8
    • Modify each layer in $Net_{UAN}^{tr}$ with input $x_i$
      to take $\hat{x}_i$ instead
end
if $update = weight$ then
    // Update parameters as weights of the neural network
    Train $Net_{UAN}^{tr}$ to optimize the parameters:
      $\Theta = \Theta \cup \{\lambda_k, \mu_k, \sigma_k^2\}_{k=1}^{K}$
end
if $update = moving\ average$ then
    // Update parameters using moving average according
     the batch
    for $k \leftarrow 1$ to $K$ do
        $\lambda_k \leftarrow m \times \lambda_k + (1 - m) \times mean(\tau_k)$
        $\mu_k \leftarrow m \times \mu_k + (1 - m) \times mean(\hat{x}^k)$
        $\sigma_k^2 \leftarrow m \times \sigma_k^2 + (1 - m) \times variance(\hat{x}^k)$
    end
end
$Net_{UAN}^{inf} = Net_{UAN}^{tr}$ // Inference UAN network with
 frozen parameters
for $i \leftarrow 1$ to $m$ do
    • Transform $\hat{x}_i = UAN(x_i)$ using Equation 8
    • $\hat{x}_i = \sum_{k=1}^{K} \frac{\tau_k(x_i)}{\sqrt{\lambda_k}} (\frac{x_i - \mu_k}{\sigma_k})$
    • Modify each layer in $Net_{UAN}^{inf}$ with input $x_i$
      to take $\hat{x}_i$ instead
end
```

IV. Experiments

In the upcoming experiments, our goal is to evaluate the performance of the Unsupervised Adaptive Normalization (UAN) method and compare it with Batch Normalization (BN) and Mixture Normalization (MN). We suggest applying UAN to tasks involving classification (Section IV-B), and domain adaptation (Section IV-C). UAN, a deep neural network layer equipped with Gaussian mixture parameters, undergoes a distinctive initialization process during training. The mean (μ) parameters are set with random values uniformly distributed between -1.0 and 1.0, establishing a varied starting point conducive to capturing central tendencies. The standard deviation (σ) parameters receive initialization with strictly positive random values within the range of 0.001 to 0.01, ensuring numerical stability and facilitating the exploration of diverse data patterns. Additionally, the mixing coefficients (λ) parameters are initialized with values spanning 0.01 to 0.99. These values are normalized to ensure their collective sum equals 1, thereby establishing a well-balanced and probabilistically sound foundation for incorporating prior knowledge within the Gaussian mixture model.

A. Datasets

The experiments in this study utilize several benchmark datasets widely recognized within the classification community:

- **CIFAR-10:** A dataset with 50000 training images and 10000 test images, each of size 32×32 pixels, distributed across 10 classes [8].
- **CIFAR-100:** Derived from the Tiny Images dataset, it consists of 50000 training images and 10000 test images of size 32×32, divided into 100 classes grouped into 20 superclasses [9].
- **Tiny ImageNet:** A reduced version of the ImageNet dataset, containing 200 classes with 500 training images and 50 test images per class [10].
- **MNIST digits:** Contains 70000 grayscale images of size 28×28 representing the 10 digits, with around 6000 training images and 1000 testing images per class [11].
- **SVHN:** A challenging dataset with over 600000 digit images, focusing on recognizing digits and numbers in natural scene images [12].

B. Unsupervised Adaptive Normalization on Classification task

1) Shallow Convolutional Neural Network: In this experimental setup, we utilize a Shallow Convolutional Neural Network (Shallow CNN) [13] architecture following the design principles outlined in the Mixture Normalization (MN) paper. The Shallow CNN comprises four convolutional layers, each activated by a Rectified Linear Unit (ReLU) [14], and subsequently normalized using a Batch Normalization (BN) layer. One notable challenge with BN arises when employing nonlinear functions (e.g., ReLU) post-activation normalization. To address this issue, we introduce Unsupervised Adaptive Normalization (UAN) in the Shallow CNN, replacing the BN layer. We demonstrate how UAN resolves this issue, improving convergence and overall performance during training.

During the training phase on CIFAR-10, CIFAR-100, and Tiny ImageNet datasets, we apply the MN method, which involves estimating a Gaussian mixture model through Maximum Likelihood Estimation (MLE) [15]. We utilize three components identified by the Expectation-Maximization (EM) algorithm [7] for MN. For a fair comparison, we also employ $K = 3$ clusters for UAN. We conduct experiments with varying learning rates from 0.001 to 0.005, use a batch size of 256, and train for 100 epochs with the AdamW optimizer [16]. To assess the efficacy of normalization methods, we substitute the third BN layer in Shallow CNN with an MN layer and replicate this process for the UAN layer.

During the training process on CIFAR-10, we captured periodic snapshots every 25 epochs while utilizing the UAN normalization. Subsequently, the model was deployed on a random CIFAR-10 batch, and the resulting activations underwent t-SNE visualization. The visualizations unveiled the formation of clusters aligning with the target classes predicted by the model. We observe an improvement in this clustering as the training, contributes to a comprehensive enhancement in performance, as depicted in Figure 2.

Figure 3 illustrates that UAN outperforms both BN and MN in terms of convergence speed. This faster convergence is evident in the improved performance on the validation dataset. Specifically, UAN shows an average accuracy increase of **2%** on CIFAR-10, **3%** on CIFAR-100, and **4%** on Tiny ImageNet. These enhancements are consistent across different class numbers (10, 100, 200) and are even more pronounced when the learning rate is raised from 0.001 to 0.005. The increasing disparity in convergence rates at higher learning rates highlights the superiority of our normalization method in effectively utilizing higher learning rates to achieve better training results. These findings align with the observed improvement in cluster formation, affirming the positive impact of the proposed normalization method on model performance.

To validate the hypothesis that Unsupervised Adaptive Normalization (UAN) substantially aids in accelerating the convergence of neural network training, we conduct the upcoming experiment using a Deep Convolutional Neural Network on the CIFAR-100 dataset.

2) Deep Convolutional Neural Network: we have shown that integrating UAN accelerates convergence and enhances performance in model using batch normalization layer as baseline. Experiments were conducted on a shallow architecture with four convolutional layers, prompting the question of whether this behavior extends to more complex architectures. To investigate that, we choose DenseNet [17], leveraging 40 and 100-layer architectures. All models undergo 200 epochs on CIFAR-100, with a batch size of 64, Nesterov's accelerated gradient [18], and initial learning rate of 0.1, reduced by a factor of 10 at 50% and 75% of training epochs. Weight decay and momentum are 10^{-4} and 0.9, respectively.

To provide a comparative viewpoint, we substitute the Batch Normalization (BN) layer in both the DensetNet-40 and DenseNet-100 architectures with layers such as MN and UAN, resulting in separate models. In the case of the MN layer, we employ 5 Gaussian components. As in the initial experiment, for a fair comparison, we designate MN components as individual clusters ($K = 5$) for models with UAN layer.

Figure 4 illustrates that UAN offers more than just accelerated training optimization—it consistently demonstrates superior generalization across various architectural setups. This advantage is particularly evident in the complex DenseNet architecture. Using DenseNet-40 on CIFAR-100, the test error for Batch Normalization (BN) and Mixture Normalization (MN) is respectively **0.38** and **0.37**, while the UAN achieves a notably lower test error of **0.36**. With the deeper DenseNet-100, the test errors for BN and MN drop respectively to **0.3** and **0.29**, while the UAN variant achieves the best test error of **0.28**.

C. Unsupervised Adaptive Normalization on Domain Adaptation

In this experimental setup, we demonstrate how the effectiveness of Unsupervised Adaptive Normalization enhancing local representations can lead to significant advancements in domain adaptation. As outlined in [19], domain adaptation involves leveraging knowledge gained by a model from a related domain, where there is an abundant amount of labeled data, to improve the model's performance in a target domain with limited labeled data. We consider two domains ($K = 2$) as two clusters: the "source domain" and the "target domain" in conjunction with AdaMatch [20]. This method combines the tasks of unsupervised domain adaptation (UDA), semi-supervised learning (SSL), and semi-supervised domain adaptation (SSDA). In UDA, we have access to a labeled dataset from the source domain and an unlabeled dataset from the target domain. The goal is to train a model capable of effectively generalizing to the target dataset. It's essential to note that the source and target datasets exhibit variations in distribution. Specifically, we use the MNIST dataset as the source dataset (cluster $k = 1$), while the target dataset is SVHN (cluster $k = 2$). Both datasets encompass various factors of variation, including texture, viewpoint, appearance, etc., and their domains, or distributions, are distinct.

AdaMatch [21] is trained from scratch using wide residual networks [22] on dataset pairs, forming the baseline model. The model training involves employing the Adam optimizer [23] with a cosine decay schedule, gradually reducing the initial learning rate initialized at 0.03. In this practical scenario, Mixture Normalization cannot be applied because we aim to tailor normalization according to domains, which may not necessarily constitute a Gaussian mixture. UAN serves as the initial layer in AdaMatch, resulting in the creation of AdaMatch model with Unsupervised Adaptive Normalization(AdaMatch+UAN). This incorporation enables the cluster identifier (source domain and target domain) to influence the image normalization process.

Table I unmistakably underscores a noteworthy improvement in validation metrics resulting from the integration of Unsupervised Adaptive Normalization. This progress is evident, translating into a substantial accuracy boost of **8.32%**. This notable enhancement significantly strengthens the performance of the AdaMatch model, contributing to a markedly accelerated convergence during training.

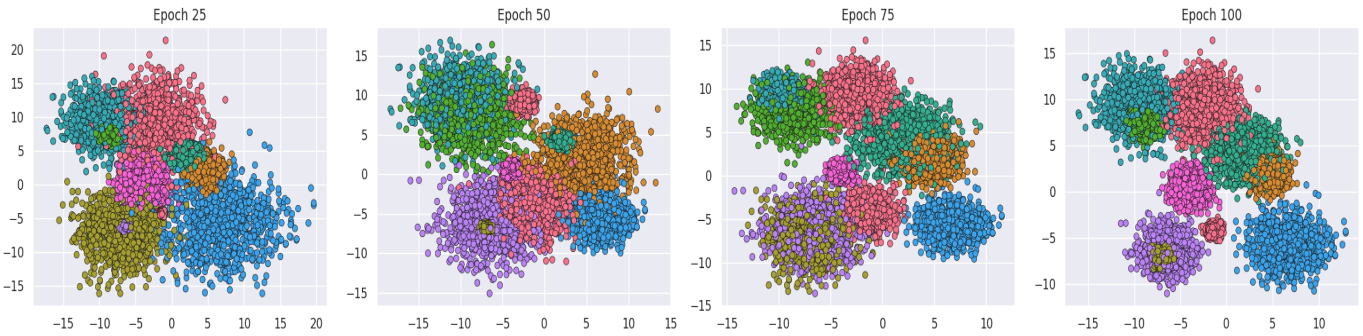
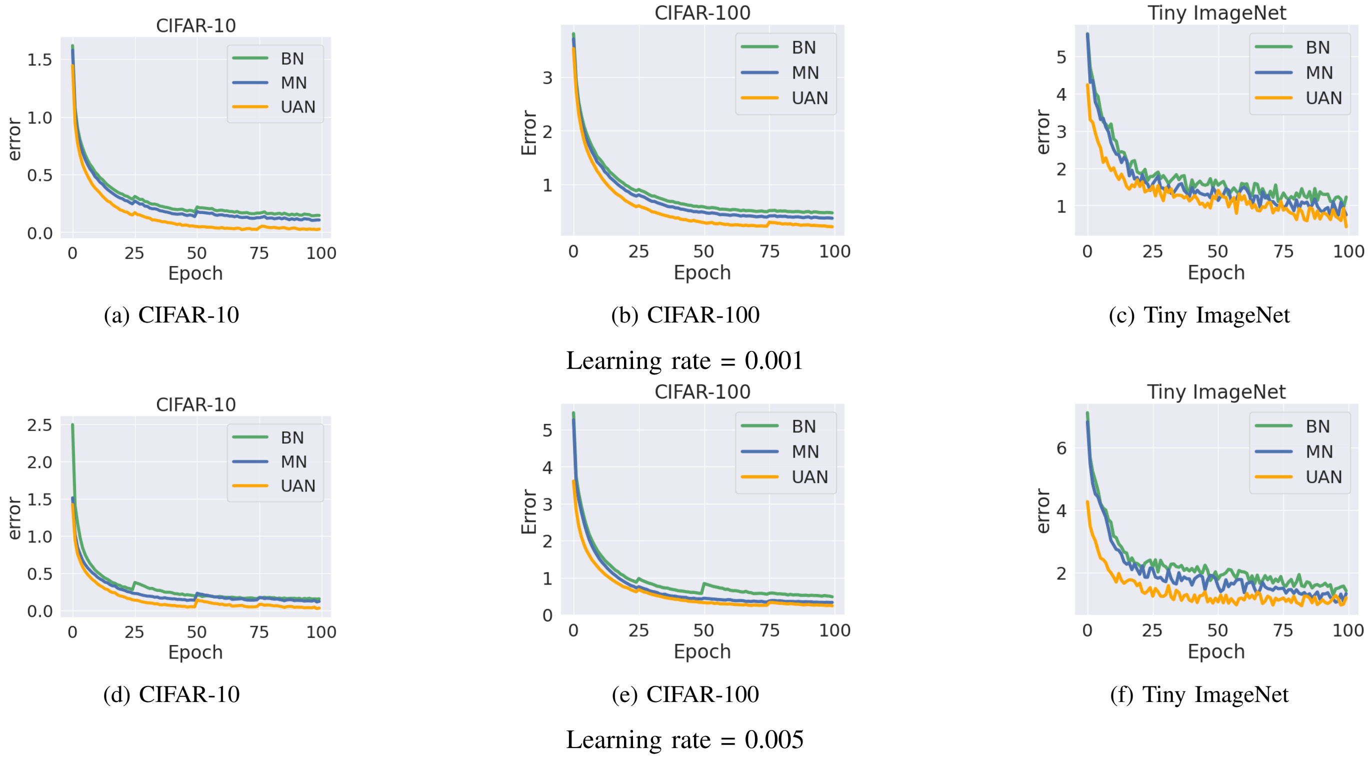


Fig. 2: t-SNE visualization of activations in latent space after 25, 50, 70 and 100 training epochs. Unsupervised Adaptive Normalization (UAN) on CIFAR-10, showing the formation and refinement of class-specific clusters over training epochs.



(a) CIFAR-10

(b) CIFAR-100

(c) Tiny ImageNet

Learning rate = 0.001

(d) CIFAR-10

(e) CIFAR-100

(f) Tiny ImageNet

Learning rate = 0.005

Fig. 3: Test error curves when the Shallow CNN architecture is trained under different learning rates.

MNIST (source domain)				
model	accuracy	precision	recall	f1-score
AdaMatch	97.36	87.33	79.39	78.09
AdaMatch+UAN	98.9	98.5	98.90	98.95
SVHN (target domain)				
model	accuracy	precision	recall	f1-score
AdaMatch	25.08	31.64	20.46	24.73
AdaMatch+UAN	33.4	43.83	40.28	42.87

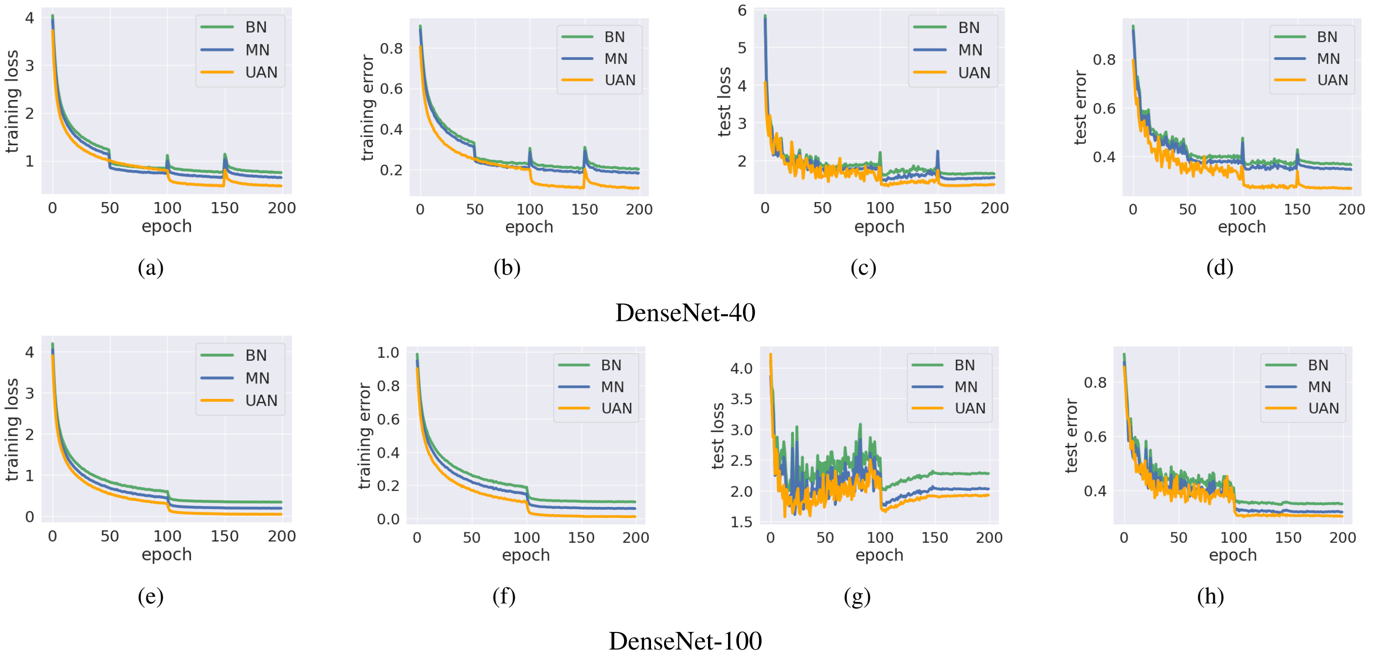
TABLE I: Comparing model performance: AdaMatch vs. AdaMatch+UAN on MNIST (source) and SVHN (target) datasets.

Figure 5 highlights the invaluable role of Unsupervised Adaptive Normalization in stabilizing the gradient throughout the training process, providing advantages for both the source and target domains. This stabilization significantly contributes to accelerated convergence and an overall improvement in model performance.

V. Computational Complexity

Compared to Batch Normalization [2] and Mixture Normalization [6], the computational overhead of adaptative normalization arises from estimating cluster parameters. Let B represent a batch of activations in $\mathbb{R}^d$, where d is the dimensionality of activations.

For Batch Normalization, mean (μ_B) and standard deviation (σ_B) are computed considering the entire batch B, leading to a computational complexity of $\mathcal{O}(d)$. Learning parameters γ



(a) (b) (c) (d)

DenseNet-40

(e) (f) (g) (h)

DenseNet-100

Fig. 4: Figures present CIFAR-100 experiments with DenseNet [17]. The graphs showcase training/test errors and cross-entropy loss for DenseNet-40 (40 layers) and DenseNet-100 (100 layers). Unsurprisingly, introducing Unsupervised Adaptive Normalization remarkably accelerates the training process, facilitating optimization. Simultaneously, it exhibits superior generalization, consistently outperforming batch normalization and mixture normalization counterparts in the error curve.

and β for each batch dimension adds a parameter learning complexity of $\mathcal{O}(d)$. Thus, the overall complexity, accounting for both computational and parameter aspects, is $\mathcal{O}(d)$ for the entire batch B.

Mixture normalization faces computational challenges in estimating Gaussian mixture model (GMM) parameters accurately. Using K-means++ for seeding, complexity for batch B with dimension d is $\mathcal{O}(d \cdot K \cdot \log(K))$. In the EM algorithm, per-iteration complexity is $\mathcal{O}(d \cdot n \cdot K)$, mainly determined by initialization, also $\mathcal{O}(d{\cdot}K{\cdot}\log(K))$. Like Batch Normalization, mean (μ_k) and standard deviation (σ_k) are computed per cluster k, $\mathcal{O}(K \cdot d)$. Parameter learning complexity (γ, β) per batch dimension is $\mathcal{O}(d)$. Overall complexity, considering both computational and parameter aspects, is $\mathcal{O}(d \cdot n \cdot K)$ for batch B.

Unsupervised Adaptive Normalization (UAN) estimates three learnable parameters for each cluster k: λ_k of dimension 1, and μ_k and σ_k of dimension d. This results in $2 \times K \times d + K$ parameters to be learned, with a complexity of $\mathcal{O}(K \cdot d)$. Unlike MN, UAN is performed in a single-stage process, and the complexity is solely associated with parameter estimation during backpropagation. A comparison is presented in Table II.

VI. Conclusion

In this paper, we introduced Unsupervised Adaptive Normalization (UAN), a novel one-stage unsupervised normalization technique for deep neural networks. UAN provides mixture components during the training of deep neural networks, tailoring them specifically for the target task. This method is efficient as it computes cluster parameters simultaneously with the neural network's weight adjustments, thus minimizing computational overhead compared to multi-stage normalization approaches or classical ones. UAN stabilizes neural network training across a range of applications, including supervised classification, and domain adaptation, proving its versatility and effectiveness in various contexts. In future work, we plan to further validate the efficacy of our approach on different context as Generative Adversarial Networks (GANs) and different types of data, particularly focusing on its adaptability to architectures that handle multimodal data.

References

[1] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2002, pp. 9–50.

[2] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

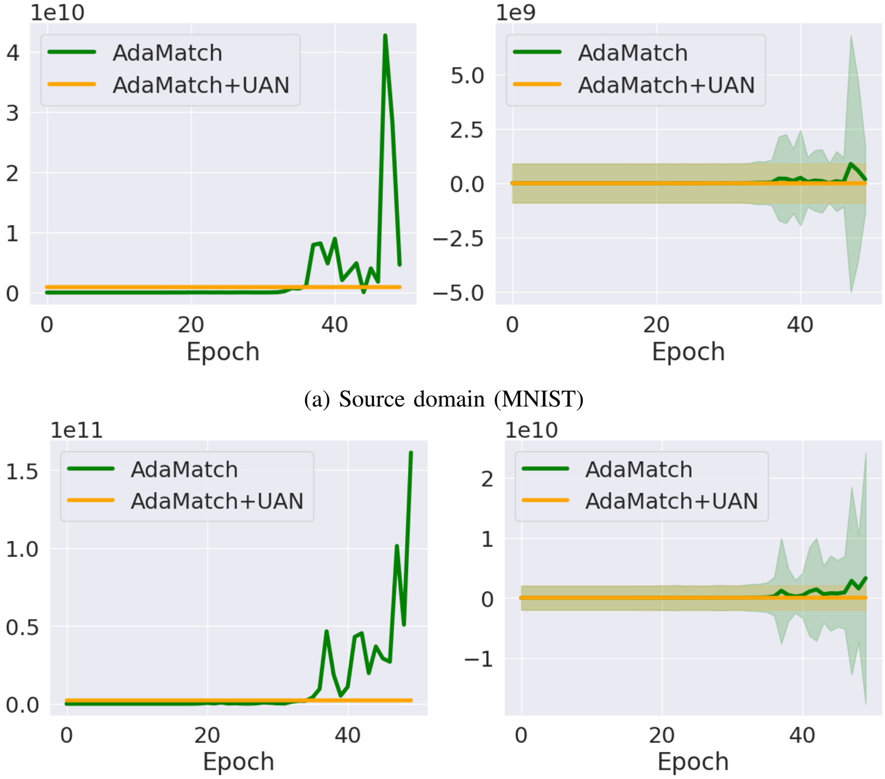
[3] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[4] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *arXiv preprint arXiv:1607.08022*, 2016.

[5] Y. Wu and K. He, "Group normalization," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.

[6] M. M. Kalayeh and M. Shah, "Training faster by separating modes of variation in batch-normalized models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 6, pp. 1483–1500, 2019.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *JOURNAL OF THE*



(a) Source domain (MNIST)

(b) Target domain (SVHN)

Fig. 5: Evolution of the gradient variance during the training of AdaMatch and AdaMatch+UAN models on the source (MNIST) and target (SVHN) domains. The figures on the left correspond to the maximum gradient variance for each epoch, while the figures on the right correspond to the average gradient variance per epoch.

Normalization Method	Computational Complexity	Parameter Complexity
Batch Normalization (BN)	$\mathcal{O}(d)$	$\mathcal{O}(d)$
Mixture Normalization (MN)	$\mathcal{O}(d \cdot n \cdot K)$	$\mathcal{O}(d)$
Unsupervised Adaptive Normalization (UAN)	0	$\mathcal{O}(K \cdot d)$

TABLE II: Complexities in Estimating Parameters for Normalization Methods. Computational Complexity corresponds to the time required for parameter estimation, while Parameter Complexity corresponds to the number of parameters to be estimated. Here, d represents the dimension of the data, K denotes the number of cluster centers, and n the number of samples.

ROYAL STATISTICAL SOCIETY, SERIES B, vol. 39, no. 1, pp. 1–38, 1977.

[8] A. Krizhevsky, V. Nair, and G. Hinton, “CIFAR-10 (canadian institute for advanced research),” 2009.

[9] ——, “CIFAR-100 (canadian institute for advanced research),” 2009.

[10] Y. Le and X. Yang, “Tiny imagenet visual recognition challenge,” *CS 231N*, vol. 7, no. 7, p. 3, 2015.

[11] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010.

[12] P. Sermanet, S. Chintala, and Y. LeCun, “Convolutional neural networks applied to house numbers digit classification,” in *Proceedings of the 21st international conference on pattern recognition (ICPR2012)*. IEEE, 2012, pp. 3288–3291.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[14] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010.

[15] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[16] I. Loshchilov and F. Hutter, “Fixing weight decay regularization in adam,” *arXiv preprint arXiv:1711.05101*, 2017.

[17] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[18] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, “Advances in optimizing recurrent networks,” in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 8624–8628.

[19] A. Farahani, S. Voghoei, K. Rasheed, and H. R. Arabnia, “A brief review of domain adaptation,” *Advances in Data Science and Information Engineering: Proceedings from ICDATA 2020 and IKE 2020*, pp. 877–894, 2021.

[20] D. Berthelot, R. Roelofs, K. Sohn, N. Carlini, and A. Kurakin, “Adamatch: A unified approach to semi-supervised learning and domain adaptation,” *arXiv preprint arXiv:2106.04732*, 2021.

[21] S. Paul, “Unifying semi-supervised learning and unsupervised domain adaptation with adamatch,” 2019, https://github.com/keras-team/keras-io/tree/master.

[22] S. Zagoruyko and N. Komodakis, “Wide residual networks,” *arXiv preprint arXiv:1605.07146*, 2016.

[23] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.