Understanding Batch-Normalization in Deep Neural Networks

A dissertation submitted in
partial fulfilment for the degree of

Master of Technology

in

Computer Science

by

Pendyala Sai Srujan
Roll no. - **CS2335**

under the supervision of

Prof. (Dr.) Sasanka Roy
Advanced Computing and Microelectronics unit (ACMU)

and

Prof. (Dr.) Shubhra Sankar Ray
Center for Soft Computing Research (CSCR)

Indian Statistical Institute, Kolkata
June, 2025

CERTIFICATE

This is to certify that the dissertation entitled **"Understanding Batch-Normalization in Deep Neural Networks"** submitted by **Pendyala Sai Srujan** to the Indian Statistical Institute, Kolkata, in partial fulfillment of the requirements for the degree of Master of Technology in Computer Science, is an authentic and genuine record of the research work carried out by the candidate under our supervision and guidance. We affirm that the dissertation has met all the necessary requirements in accordance with the regulations of this institute.

Sasanka Roy .11/06/25

Prof. (Dr.) Sasanka Roy
ACMU
Indian Statistical Institute
Kolkata - 700108
India

Shubhra Sankar Ray 11.6.25

Prof. (Dr.) Shubhra Sankar Ray
CSCR Unit
Indian Statistical Institute
Kolkata - 700108
India

Acknowledgment

I extend my sincere gratitude to Prof. (Dr.) Sasanka Roy and Prof. (Dr.) Shubhra Sankar Ray, my supervisors at the Indian Statistical Institute, Kolkata for their invaluable guidance, immense support and inspiration. Their profound knowledge and creative suggestions have taught me a great deal in every subject and have shown me how to conduct solid research.

I would like to sincerely thank Joginder Singh, Senior Research Fellow at the Indian Statistical Institute, Kolkata for his continuous assistance in guiding me throughout my dissertation period. His consistent provision of ideas and unwavering support have been instrumental to the completion of this project.

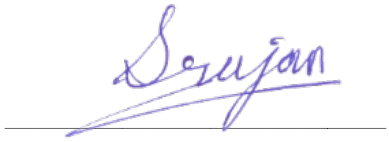
I am deeply grateful to all the teachers at the Indian Statistical Institute, Kolkata for their invaluable advice, insights, and instruction, which provided a crucial perspective to my research. Special thanks go to the Mentor committee for their constant support and mentoring.

Finally, I would like to express my sincere gratitude to my parents and brother for their unwavering support. I also extend my sincere appreciation to all my colleagues for their continuous encouragement. I am thankful to everyone who has contributed to my growth and success, even if I have inadvertently missed mentioning them in the above list.

Declaration

I, **Pendyala Sai Srujan**, with Roll No. **CS2335**, hereby declare that the material presented in the dissertation titled **Understanding Batch-Normalization in Deep Neural Networks** represents original work carried out by me for the degree of **Master of Technology in Computer Science** at the **Indian Statistical Institute, Kolkata**.

Furthermore, I affirm that no sections of this report have been sourced or copied from external references without proper attribution. I am aware that any instances of plagiarism or the use of unacknowledged materials from third parties will be treated with the utmost seriousness and consequences.



Pendyala Sai Srujan
M.Tech (CS), CS2335
Indian Statistical Institute

Abstract

Batch Normalization (BN) is a commonly used technique in various deep learning architectures for tasks such as image classification and object detection. It stabilizes and accelerates training by normalizing the activations of intermediate layers using mean and variance of the batch, allowing the use of higher learning rates and often improving generalization through implicit regularization.

During inference, BN uses running estimates of batch statistics accumulated during training. However, if individual batches are not representative of the overall data distribution, these accumulated statistics may not accurately approximate the population statistics. This discrepancy can lead to a phenomenon known as **estimation shift**, which impairs the model's generalization performance.

In this project, we study the behavior of estimation shift in deep learning models using BN and explore techniques to mitigate its effects. Specifically, we introduce **dynamicity** in the momentum parameter of BN layer (DMBN) while computing exponential moving averages and evaluate its impact under various architectural configurations. We use MNIST, FashionMNIST, and CIFAR-10/100 datasets to train and test both simple Deep Neural Networks (DNNs) as well as deeper Convolutional Neural Networks (CNNs) such as ResNet-50.

Our experiments are conducted in two phases: first, by varying the static momentum parameter across different values, and second, by introducing layer-wise dynamic momentum where each layer is assigned the momentum (or equivalently, β) that minimizes estimation shift. The performance of the proposed method, DMBN, is evaluated using various performance metrics such as sensitivity, specificity, accuracy, and F-score. The DMBN is compared with existing BN-BFN method and is observed to be performing better in most of cases. For example, for fashionMNIST data, the accuracy values achieved by DMBN and BN-BFN are 0.889 and 0.853, respectively.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Batch Normalization (BN) is introduced by Ioffe in 2015 to stabilize and accelerate the training of the neural networks by mitigating *internal covariate shift (ICS)* [1]. ICS is a phenomenon wherein the input distribution to each layer varies as the training parameters of the previous layers evolve during training. BN reduces the ICS by normalizing the each layers input such that they have zero mean and unit variance across the batch. This normalization helps in faster convergence, smoother optimization landscapes, and better regularization, allowing for higher learning rates and reduced sensitivity to initialization.

BN consists of several key components that enable its functionality:

- **Batch Statistics**: For each activation channel, the mean and variance are computed over the current batch.
- **Exponential Moving Average (EMA)**: During training, an EMA of the batch statistics is maintained to estimate the population mean and variance used during inference.
- **Learnable Parameters γ and δ**: γ and δ are used to scale and shift, respectively, the normalized values to preserve the network's expressive power.
- **Momentum Parameter**: This controls the smoothing of the EMA updates; higher momentum means slower adaptation to new data, and lower momentum results in faster updates.

Despite its success, BN is known to perform poorly when batch sizes are small or when batches don't represent the population well. In such cases, the batch statistics become unreliable, and the EMA used for inference becomes misaligned with the true distribution of activations, leading to a phenomenon known as *estimation shift*.

Estimation shift refers to the discrepancy between the population statistics estimated during training and the actual statistics encountered during testing. This shift becomes particularly evident when the batch statistics used to update the EMA are noisy or unrepresentative. When the statistics used at inference time deviate significantly from those encountered during training, the normalization applied by BN may distort rather than stabilize the activations, resulting in degraded model generalization.

The degree of estimation shift can be quantified using the L_2 norm between the estimated statistics and the actual test statistics at each layer. Using the concept of *Estimation Shift Magnitude (ESM)*, this notion of shift is quantitatively measured. Empirical analysis showed that this shift accumulates across layers in a deep network. Even in full-batch training scenarios estimation shift persisted, indicating a deeper issue in the estimation of these values by BN [2].

To address certain limitations of BN—especially its dependency on batch statistics—researchers introduced *Batch-Free Normalization* techniques. These methods were originally motivated by practical scenarios where Batch Normalization fails to perform effectively, such as:

- **Small Batch Training:** Common in tasks like object detection or video processing, where GPU memory constraints force batch sizes to be small.
- **Sequential or Online Learning:** Where data is streamed or processed one sample at a time.
- **Domain Adaptation and Non-i.i.d. Data:** Where the assumption of uniform data distribution across batches no longer holds.

Batch-Free Normalization methods—such as Group Normalization (GN) [3], Layer Normalization (LN) [4], and Instance Normalization (IN) [5]—perform normalization over dimensions other than the batch. For example, GN normalizes over groups of channels, LN normalizes over the full feature vector per instance, and IN normalizes each feature map separately. Because these techniques do not rely on batch statistics, they are inherently more stable in the settings listed above.

In the context of estimation shift, it was shown that introducing GN into alternating layers of deep CNNs—effectively creating a hybrid normalization scheme—significantly reduced the estimation shift and improved generalization performance. This suggests that reducing dependency on batch-level statistics can be an effective way to mitigate inference-time discrepancies [2].

Problem Statement

While replacing BN with batch-free methods mitigates estimation shift and thereby improving accuracy, it also changes the underlying dynamics and benefits that BN offers—such as faster training and implicit regularization through noise in batch statistics. Therefore, a natural question arises: *Can we achieve higher accuracy while preserving the advantages of the standard BN framework?*

This project explores whether the use of a **dynamic momentum** parameter—one that changes adaptively over the course of training or based on certain criteria—can improve the model's performance without replacing Batch Normalization with other normalization techniques. Our hypothesis is:

> *Introducing dynamic momentum in Batch Normalization should improve the model's performance.*

Our approach maintains the core principles of BN while modifying only the update behavior of the EMA. This preserves compatibility with existing architectures and optimizers while potentially enhancing generalization performance—especially. Through this study, we aim to contribute to a more robust and adaptable version of Batch Normalization that retains its training benefits while addressing a key source of test-time degradation.

Chapter 2

Related Work

Batch Normalization (BN) has played a pivotal role in stabilizing and expediting training of the neural networks. It mitigates the issue of internal covariate shift by normalizing intermediate layer inputs using mini-batch statistics. This stabilization allows for higher learning rates and improves convergence. To preserve the expressive power of the network, trainable scale and shift parameters (γ, δ) are introduced [1]. During inference, population estimates are approximated via Exponential Moving Averages (EMA) maintained over training iterations. Moreover, the noise introduced by using mini-batch statistics acts as an implicit regularizer, aiding generalization in large models [1].

Subsequent studies revealed deeper optimization benefits of BN. In particular, it was shown to smoothen the optimization landscape, making gradients more predictable and reducing the chances of vanishing or exploding gradients. This contributes further to BN's success in enabling deeper architectures and faster training [6].

Despite these strengths, BN suffers from a phenomenon termed *estimation shift*, wherein discrepancies arise between training-time (mini-batch based) and test-time (EMA-based) statistics. This shift can degrade inference-time performance. The shift was quantitatively assessed using the L_2 norm between the estimated and true test statistics across layers—referred to as *Estimation Shift Magnitude (ESM)*. Notably, ESM was found to accumulate with network depth and persisted even under full-batch training, indicating that the issue stems from the very nature of BN's statistic estimation rather than batch size alone [2].

To mitigate this, the authors proposed a hybrid normalization framework that interleaves Batch Normalization layers with batch-free normalization (BFN) layers

such as Group Normalization (GN). Since BFN methods compute statistics independent of the mini-batch, they are less susceptible to estimation shift. This alternating structure successfully suppressed ESM while preserving BN's regularization and convergence benefits, resulting in improved test-time accuracy and robustness [2].

Prior to this, several efforts were made to stabilize BN in small-batch or noisy-data regimes. Batch Renormalization introduced a corrective affine transformation during training to align batch statistics more closely with the accumulated statistics. This approach aimed to reduce the dependency of activations on other samples in the mini-batch, thereby narrowing the training-inference gap [7].

Other strategies sought to adapt BN's momentum parameter dynamically during training. By tuning momentum based on batch size or iteration count, these methods aimed to maintain smooth updates to the running statistics, particularly under small or unstable batches. While such modifications reduced some statistical noise, they remained within the batch-dependent framework [8].

To overcome the fundamental limitations of batch-dependence, several batch-free normalization techniques have been proposed. These methods avoid using batch dimensions when computing normalization statistics, making them suitable for small-batch or online learning scenarios.

Layer Normalization (LN) was introduced to address limitations of Batch Normalization in tasks where batch size is small or variable, especially in recurrent neural networks (RNNs). Unlike BN, which computes statistics across the batch, LN calculates the mean and variance across all features within a single training example. This allows the normalization process to be independent of the batch composition, making it more stable in sequence modeling. LN ensures consistent behavior between training and inference since it does not rely on batch-wide statistics. It has been shown to stabilize hidden state dynamics and improve convergence speed in RNNs and Transformer-based models [4].

Instance Normalization (IN) was designed with style transfer in mind. It normalizes each channel of each sample independently by computing statistics over only the spatial dimensions. This removes instance-specific contrast and appearance, effectively decoupling style from content. In the context of neural style transfer and image generation tasks, replacing BN with IN resulted in significantly improved stylization quality [5]. IN is now a widely adopted technique in generative adversarial networks (GANs) and image synthesis applications due to its ability to isolate style

information from the content.

Group Normalization (GN) was introduced to address BN's dependency on batch size, which poses challenges in tasks like object detection and video processing. GN divides the channels into multiple groups and computes mean and variance within every group, normalizing features independently of the batch. It effectively bridges the gap between LN and IN by allowing spatial aggregation within subgroups of channels while maintaining independence from the batch. GN has shown robust accuracy across a wide range of batch sizes and is especially useful in memory-constrained environments where large batch sizes are infeasible [3].

These batch-free alternatives offer stable normalization across diverse training conditions, though they sometimes underperform BN in large-batch regimes.

Chapter 3

Preliminaries

3.1 Normalization techniques in Deep Neural Networks

Batch Normalization (BN)

Batch Normalization (BN) is a technique to normalize the activations of each layer in deep neural networks using batch statistics during training.

Let $\mathbf{x} \in \mathbb{R}^d$ be the input to a layer for a given training example. Given a batch of m samples, BN transformation for feature dimension j is defined as:

$$\hat{x}_j = \mathrm{BN}_{\mathrm{train}}(x_j) = \frac{x_j - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}, \quad j = 1, 2, \ldots, d, \tag{3.1}$$

where:

- $\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$ is the batch mean for feature j,
- $\sigma_j^2 = \frac{1}{m} \sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2$ is the batch variance,
- ϵ is a small constant to avoid division by zero.

During inference, however, batch statistics are not available. Instead, BN uses running averages of the training means and variances, denoted as $\tilde{\mu}_j$ and $\tilde{\sigma}_j^2$:

$$\hat{x}_j = \mathrm{BN}_{\mathrm{test}}(x_j) = \frac{x_j - \tilde{\mu}_j}{\sqrt{\tilde{\sigma}_j^2 + \epsilon}}, \quad j = 1, 2, \ldots, d. \tag{3.2}$$

These population statistics are updated during training using an exponential moving average (EMA):

$$\begin{aligned} \hat{\mu}^{(t)} &= (1-\beta)\hat{\mu}^{(t-1)} + \beta\mu^{(t)}, \\ (\hat{\sigma}^{(t)})^2 &= (1-\beta)(\hat{\sigma}^{(t-1)})^2 + \beta(\sigma^{(t)})^2, \end{aligned} \tag{3.3}$$

Equation (3.3) defines the exponentially weighted moving averages (EMA) used to estimate the population mean and variance from mini-batch statistics during training. These estimated values, $\hat{\mu}^{(t)}$ and $(\hat{\sigma}^{(t)})^2$, are later used for normalization during inference to ensure consistency and determinism when batch statistics are not available.

The terms in the equation are interpreted as follows

- $\hat{\mu}^{(t)}$ and $(\hat{\sigma}^{(t)})^2$: Running estimates of the population mean and variance at training step t.
- $\hat{\mu}^{(t-1)}$ and $(\hat{\sigma}^{(t-1)})^2$: Previously accumulated values from step $t-1$.
- $\mu^{(t)}$ and $\sigma^{(t)})^2$: Batch mean and variance computed from the most recent mini-batch at step t.
- β: Momentum parameter (typically 0.1 or 0.01) that controls the weight given to the new batch statistics. A larger β means the model adapts more quickly to recent data, while a smaller β represents slower updates.

This momentum-based update allows the model to build a stable estimate of the population statistics over time, which is especially important during inference when batch data is not used.

BN introduces a discrepancy between training and inference due to this switch in normalization statistics, which can degrade performance—especially for small batch sizes or in recurrent models.

Batch-Free Normalization (BFN)

Batch-Free Normalization methods eliminate dependence on the batch dimension, thereby applying the same normalization procedure during both training and inference.

Layer Normalization (LN) is a representative BFN method, which computes normalization statistics across the feature dimension for each individual sample:

$$\hat{x}_j = \mathrm{LN}(x_j) = \frac{x_j - \mu}{\sqrt{\sigma^2 + \epsilon}}, \quad j = 1, 2, \ldots, d, \tag{3.4}$$

where:

- $\mu = \frac{1}{d}\sum_{j=1}^{d} x_j$, the mean over features of a single sample,
- $\sigma^2 = \frac{1}{d}\sum_{j=1}^{d}(x_j - \mu)^2$, the feature-wise variance for that sample.

Group Normalization (GN) divides features into groups and applies Layer Normalization (LN) within each group. **Group Normalization (GN)** divides features into groups and applies Layer Normalization (LN) within each group, making it more flexible and better suited to convolutional layers with small batch sizes.

While BFN methods avoid the training–inference mismatch and perform better for small-batch settings, they generally underperform BN in large-batch regimes or standard convolutional architectures.

Normalization Axes in Convolutional Neural Networks

The previous descriptions focus on vectorized features $\mathbf{x} \in \mathbb{R}^d$ as commonly used in Deep Neural Networks (DNNs). In Convolutional Neural Networks (CNNs), however, inputs and activations are typically 4D tensors of shape $N \times C \times H \times W$, representing:

- N: batch size,
- C: number of channels,
- H, W: spatial dimensions (height and width).

Normalization techniques differ in which axes they compute statistics over:

- **BatchNorm (BN)**: Normalizes across batch, height, and width for each channel [1]:

$$\mu_c = \frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{n,c,h,w}, \quad \sigma_c^2 = \frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{n,c,h,w} - \mu_c)^2$$

- **InstanceNorm (IN)**: Normalizes across spatial dimensions H, W for each sample and channel [5]:

$$\mu_{n,c} = \frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{n,c,h,w}, \quad \sigma_{n,c}^2 = \frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{n,c,h,w} - \mu_{n,c})^2$$

- **LayerNorm (LN)**: Normalizes across all features (channels and spatial dimensions) within each sample [4]:

$$\mu_n = \frac{1}{CHW}\sum_{c=1}^{C}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{n,c,h,w}, \quad \sigma_n^2 = \frac{1}{CHW}\sum_{c=1}^{C}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{n,c,h,w} - \mu_n)^2$$

- **GroupNorm (GN)**: Divides channels into G groups and normalizes over $(C/G) \cdot H \cdot W$ elements within each group [3]:

$$\mu_{n,g} = \frac{1}{(C/G)\cdot H \cdot W}\sum_{c\in g}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{n,c,h,w}, \quad \sigma_{n,g}^2 = \frac{1}{(C/G)\cdot H \cdot W}\sum_{c\in g}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{n,c,h,w} - \mu_{n,g})^2$$

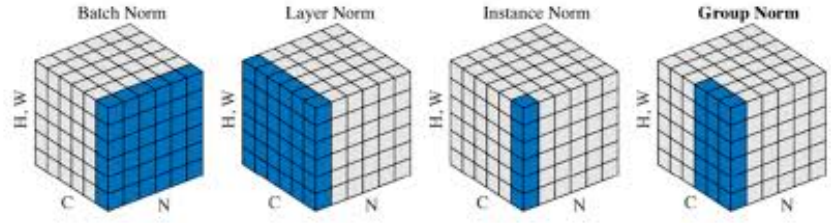


Figure 3.1: Comparison of normalization axes in BN, IN, LN, and GN [3].

This spatial-axis view offers practical intuition when applying these normalization schemes in CNNs. For instance, GroupNorm has become popular in detection/segmentation pipelines where batch size is small, while BatchNorm remains preferred for classification tasks with large batches.

Evaluation Metrics

To quantitatively assess model performance in multi-class classification tasks, we use the following evaluation metrics. For a given class c, we define:

- TP_c: True Positives – samples of class c correctly predicted as c,
- TN_c: True Negatives – samples not of class c correctly predicted as not c,
- FP_c: False Positives – samples not of class c incorrectly predicted as c,
- FN_c: False Negatives – samples of class c incorrectly predicted as another class.

To compute aggregate metrics across all K classes (e.g., $K = 100$ for CIFAR-100), we use macro-averaging, which treats each class equally:

- **Accuracy**: Overall proportion of correctly classified samples.

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(y_i = \hat{y}_i)$$

where N is the total number of samples, y_i is the true label, and $\hat{y}_i$ is the predicted label.

- **Sensitivity (Recall)**: Average recall across all classes.

$$\text{Sensitivity} = \frac{1}{K}\sum_{c=1}^{K} \frac{TP_c}{TP_c + FN_c}$$

- **Specificity**: Average specificity across all classes.

$$\text{Specificity} = \frac{1}{K}\sum_{c=1}^{K} \frac{TN_c}{TN_c + FP_c}$$

- **F1-Score**: Average F1-score across all classes.

$$\text{F1-Score} = \frac{1}{K}\sum_{c=1}^{K} \frac{2TP_c}{2TP_c + FP_c + FN_c}$$

These macro-averaged metrics provide a comprehensive and class-balanced evaluation of performance.

3.2 Models

Initial experiments were conducted using a 20-layer MLP [9] on MNIST and FashionMNIST datasets. However, as network depth increases, training becomes difficult due to vanishing gradients in the earlier layers.

To address this, skip connections [10] are introduced in ResNet architecture, allowing gradients to flow directly across layers in complex models. This work uses the ResNet-50 architecture to support deep learning tasks on larger datasets i.e. CIFAR-10 and CIFAR-100. ResNet architecture is shown in Fig. 3.2.

Activation Functions

In this work, the Rectified Linear Unit (ReLU) activation function is used in all experiments to maintain consistency with the experimental setup presented in [2].

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3.2: ResNet architectures from [10]

ReLU is commonly used in deep learning because it is simple to compute and helps prevent the vanishing gradient problem.

For completeness, we also present the mathematical formulations of other commonly used activation functions in the literature—Sigmoid and Tanh—as they exhibit different properties in terms of non-linearity, saturation, and gradient dynamics. The functions and their respective derivatives are summarized in Table 3.1, and their behavior is visualized in Figure 3.3.

Table 3.1: Common Activation Functions and Their Derivatives

Function	**Definition**	**Derivative**
Sigmoid	$f(x) = \dfrac{1}{1+e^{-x}}$	$f'(x) = f(x)\,(1-f(x))$
Tanh	$f(x) = \dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$f'(x) = 1-f(x)^2$
ReLU	$f(x) = \max(0,x)$	$f'(x) = \begin{cases} 0 & \text{if } x<0 \\ 1 & \text{if } x>0 \\ \text{undefined} & \text{if } x=0 \end{cases}$

3.3 Estimation Shift of Batch Normalization

In relation to the exponential moving average (EMA) technique for estimating the population statistics in Batch Normalization (BN), as already defined in Equa-

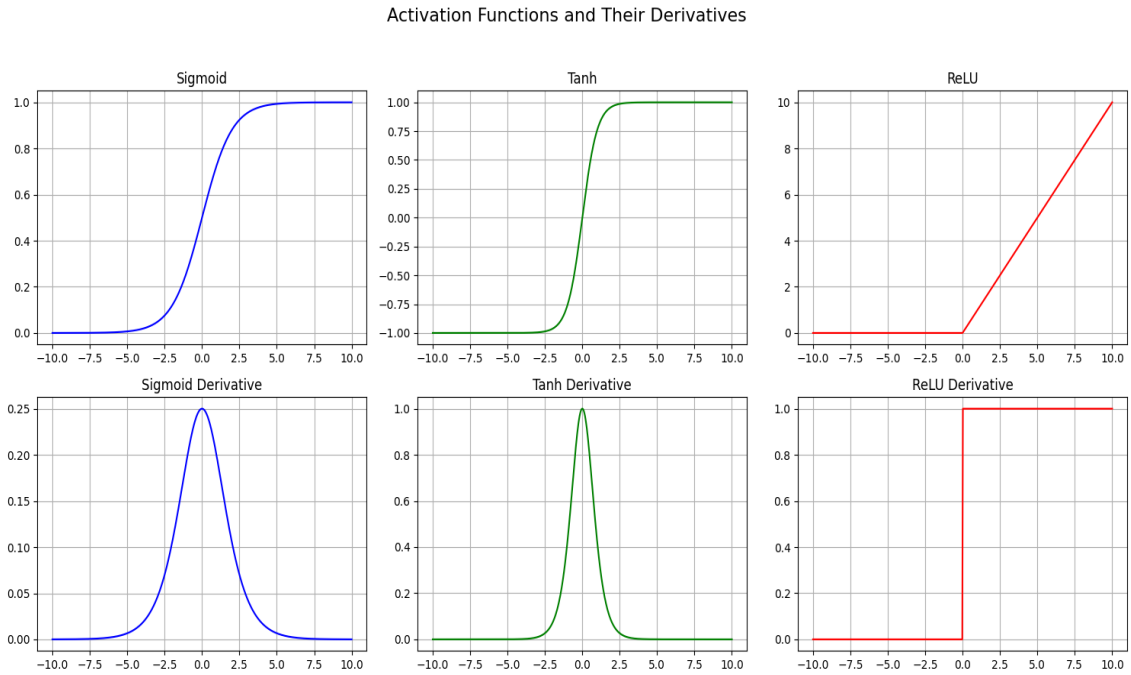


Figure 3.3: Activation functions and their derivatives: Sigmoid, Tanh and ReLU

tion 3.3, the following are the two important definitions that characterize the estimation shift issue in BN [2].

Definition 1. Let $\tilde{F}_{\psi,\tilde{\theta}}(\cdot)$ denote the trained model on a training set $\mathcal{S}$. Given a test set $\mathcal{S}'$, the expected population statistics of Batch Normalization (BN) are defined as $\{\tilde{\mu}, \tilde{\sigma}^2\}$, where $\tilde{\mu}$ and $\tilde{\sigma}^2$ are the expected mean and variance, respectively, of the BN input $X = F^{\text{pre}}_{\tilde{\theta}}(\mathcal{S}')$ before normalization.

Definition 2. Let $\hat{\mu}$ and $\hat{\sigma}^2$ be the estimated statistics obtained via running averages during training. The *Estimation Shift Magnitude* (ESM) is defined as the L_2-distance between the expected and estimated statistics:

$$\text{ESM}_\mu = \|\hat{\mu} - \tilde{\mu}\|_2, \quad \text{ESM}_\sigma = \left\| \sqrt{\hat{\sigma}^2} - \sqrt{\tilde{\sigma}^2} \right\|_2 .$$

This quantifies the mismatch between what BN expects at test time and what it actually sees, a phenomenon the authors refer to as "estimation shift" [2].

Chapter 4

Methodology

In this chapter, the details of the datasets used for the experimentation, experimental design, models used, normalization strategies implemented and evaluation metrics are discussed. The primary objective is to assess the behavior of different Batch Normalization (BN) strategies in Deep Neural Networks as well as Convolutional Neural Networks.

To that end, multiple variants of BN were evaluated on image classification tasks across datasets of increasing complexity: MNIST, FashionMNIST, CIFAR-10, and CIFAR-100. Models ranged from fully-connected deep networks to state-of-the-art convolutional architectures. Quantitative evaluation was performed using common classification metrics.

4.1 Datasets

This study uses the MNIST, FashionMNIST, CIFAR-10, and CIFAR-100 datasets to train and evaluate neural network models.

MNIST

The MNIST dataset [11] (Modified National Institute of Standards and Technology database) contains grayscale images of handwritten digits ranging from 0 to 9. A summary of the dataset is provided in Table 4.1.

FashionMNIST

FashionMNIST [12] includes grayscale images of fashion items such as T-shirts, trousers, etc. Table 4.2 presents the dataset details.

Table 4.1: MNIST Dataset Summary

Property	Value
Image Size	28×28
Image Type	Grayscale
Number of Classes	10 (Digits 0–9)
Training Samples	60,000
Test Samples	10,000
Data Range	[0, 1] (normalized)

Table 4.2: FashionMNIST Dataset Summary

Property	Value
Image Size	28×28
Image Type	Grayscale
Number of Classes	10 (e.g., T-shirt, Trouser, Coat, etc.)
Training Samples	60,000
Test Samples	10,000
Data Range	[0, 1] (normalized)

CIFAR-10

The CIFAR-10 [13] dataset contains RGB images across 10 object categories. Table 4.3 summarizes the dataset characteristics.

Table 4.3: CIFAR-10 Dataset Summary

Property	Value
Image Size	32×32
Image Type	RGB
Number of Classes	10 (e.g., Airplane, Cat, Ship, etc.)
Training Samples	50,000
Test Samples	10,000
Data Range	[0, 255] (normalized in preprocessing)

CIFAR-100

CIFAR-100 [13] is a more fine-grained version of CIFAR-10 with 100 classes, each having fewer examples. The dataset summary is given in Table 4.4.

4.2 Deep Learning Architectures

MLP Architecture for MNIST and FashionMNIST

A deep Multi-Layer Perceptron (MLP) model was designed with 20 hidden layers, each containing 128 neurons and ReLU activation. The output layer consists of 10 neurons for classification.

Table 4.4: CIFAR-100 Dataset Summary

Property	**Value**
Image Size	32×32
Image Type	RGB
Number of Classes	100
Training Samples	50,000
Test Samples	10,000
Samples per Class	500 (train), 100 (test)
Data Range	[0, 255] (normalized in preprocessing)

This architecture aligns with the experiment designed as a motivation to introduce batch-free Normalization into the network. This also helps in comparing the impact of dynamicity with the proposed BN-BFN approach (Replacing BN with BFN at alternate layers) [2].

We introduce dynamicity at each Batch Normalization layer in the network choosing respective beta that minimizes Estimaion Shift Magnitude values 3.3.

ResNet-50 Architecture for CIFAR-10 and CIFAR-100

For CIFAR-10 and CIFAR-100, we utilized a modified ResNet-50 model. The model was adapted to take input images of size 32×32. Analogous to the experiment on Deep Neural Networks Dynamicity is introduced at each Batch Normalization layer [14, 15].

4.3 Normalization Strategies

We preformed our analysis with three normalization strategies: Static Batch Normalization, Dynamic Momentum Batch Normalization, and BN-BFN Alternation. Each method is described in detail below.

Static Batch Normalization (Vanilla BN)

In this conventional approach, Batch Normalization is applied with a fixed momentum parameter β across all layers and epochs. Reiterating the EMA updation formula 3.3, during training, the running estimates of the mean and variance for each layer are updated using an EMA as follows:

$$\mu_{\text{running}}^{(t)} = (1 - \beta) \cdot \mu_{\text{running}}^{(t-1)} + \beta \cdot \mu_{\text{batch}}^{(t)}$$

$$\sigma^2_{\text{running}}{}^{(t)} = (1-\beta)\cdot\sigma^2_{\text{running}}{}^{(t-1)} + \beta\cdot\sigma^2_{\text{batch}}{}^{(t)}$$

Here, $\mu^{(t)}_{\text{batch}}$ and $\sigma^2_{\text{batch}}{}^{(t)}$ are computed from the mini-batch at iteration t. These running statistics are used during inference, where batch-wise statistics are unavailable.

We tested static values of β from 0.1 to 0.9 (in steps of 0.1) to evaluate how momentum affects estimation the model's performance.

Dynamic Momentum Batch Normalization

The momentum parameter β controls the weight given to the current batch statistics: a high β allows the running statistics to adapt more quickly to the current batch (less inertia), while a low β places more emphasis on past statistics, leading to slower adaptation.

While static momentum can work well in practice, we also wanted to test how introducing dynamicity affects the model's performance. To investigate the benefit of adapting momentum dynamically, we implemented a strategy where, for each layer l and epoch e, the value of β that minimizes the Expected Squared Mean (ESM) is selected.

Reiterating the formula presented for ESM calculation 3.3:

$$\text{ESM}_\mu = \|\mu_{\text{estimated}} - \mu_{\text{expected}}\|_2, \quad \text{ESM}_\sigma = \left\| \sqrt{\sigma^2_{\text{estimated}}} - \sqrt{\sigma^2_{\text{expected}}} \right\|_2.$$

Considering the ESM formula for a given layer l and epoch e, we have:

$$\text{ESM}^{(l,e)}_\mu = \|\mu^{(l,e)}_{\text{estimated}} - \mu^{(l,e)}_{\text{expected}}\|_2, \quad \text{ESM}^{(l,e)}_\sigma = \left\| \sqrt{\sigma^2_{\text{estimated}}{}^{(l,e)}} - \sqrt{\sigma^2_{\text{expected}}{}^{(l,e)}} \right\|_2.$$

The optimal $\beta^*_{l,e}$ for each layer and epoch is determined by:

$$\beta^*_{l,e} = \arg\min_{\beta\in\{0.1,0.2,\ldots,0.9\}} \text{ESM}^{(l,e)}_\mu(\beta) \tag{4.1}$$

This layer-wise, epoch-wise dynamic update helps align running estimates more closely with current batch statistics, mitigating estimation shift over time. The dynamically selected β^* values were tracked and visualized to understand learning dynamics across layers.

BN-BFN Alternation

This strategy seeks to counteract the negative impact of estimation shift by introducing **Batch-Free Normalization (BFN)** into the network. In Multi-Layer Perceptrons (MLP), alternate Batch Normalization (BN) layers are replaced with BFN layers, reducing reliance on batch statistics and mitigating cumulative estimation errors across depth.

The ResNet-50 architecture contains **16 bottleneck blocks** in total. In [2], in each bottleneck of ResNet architecture, there exists three BN layers say BN1 (BN after first convolution layer), BN2 (BN after second convolution layer), and BN3 (BN after third convolution layer). They proposed three variants of ResNet-50 namely **XBNBlock-P1**, **XBNBlock-P2**, and **XBNBlock-P3**. In XBNBlock-P1, the first BN layer in each bottleneck block—i.e., the BN layer following the first convolution—is replaced with a BFN layer, while the second and third BN layers remain unchanged. Similarly in XBNBlock-P2 and XBNBlock-P3 the corresponding BN layer is replaced with BFN layer while keeping others unchanged. XBNBlock-P2 achieved highest accuracy among the three variants [2].

In our experiments, we used **BN-BFN alternating** and **XBNBlock-P2** respectively for MLP and ResNet-50 architectures, as described in the paper. This strategy was included for comparative analysis against our static and dynamic BN variants.

4.4 Training and Evaluation Procedure

All models were trained using Stochastic Gradient Descent (SGD) with a momentum of 0.9 and an initial learning rate of 0.1 for a total of 50 epochs. Two training regimes were considered: full-batch training and mini-batch training, designed to capture different statistical behaviors of Batch Normalization.

Full-Batch Training: In this setting, the entire test set of MNIST or Fashion-MNIST was used both as the training and test set. This aligns with the experimental setup used in [2].

Mini-Batch Training: Here, the standard training set was split into 80% training and 20% validation subsets. The validation subset was exclusively used for calculating the Expected Squared Mean (ESM) to guide the dynamic selection of momentum β. The final model performance was evaluated on the untouched official test set of MNIST or FashionMNIST. A batch size of 100 was used for training

phase.

The **ESM metric** was computed at each BatchNorm layer after every epoch, using the validation set (in mini-batch training) or the same dataset (in full-batch training), to identify the optimal β values that minimize the estimation shift for each layer.

Evaluation Metrics

To comprehensively assess classification performance, we use the following evaluation metrics, each formally defined in Section 3.1: accuracy, sensitivity (recall), specificity, and F1-score. These metrics provide a balanced view of the model's ability to correctly classify positive and negative instances, particularly in the presence of class imbalance.

Chapter 5

Results

We begin by illustrating how the optimal β values evolve over training for a specific layer in the network. Figure 5.1 shows the evolution of the "best" β values—those minimizing the Expected Squared Mean (ESM)—for the 18^{th} layer of the Multi-Layer Perceptron (MLP) trained on the FashionMNIST dataset. The x-axis denotes training epochs, while the y-axis represents the selected β value at each epoch.

This visualization highlights that the optimal momentum is neither fixed nor monotonic. Instead, the ideal β fluctuates across epochs, reflecting the non-stationary nature of feature distributions in deep networks. Such behavior motivates our approach of dynamically or adaptively tuning momentum, rather than relying on a static value.

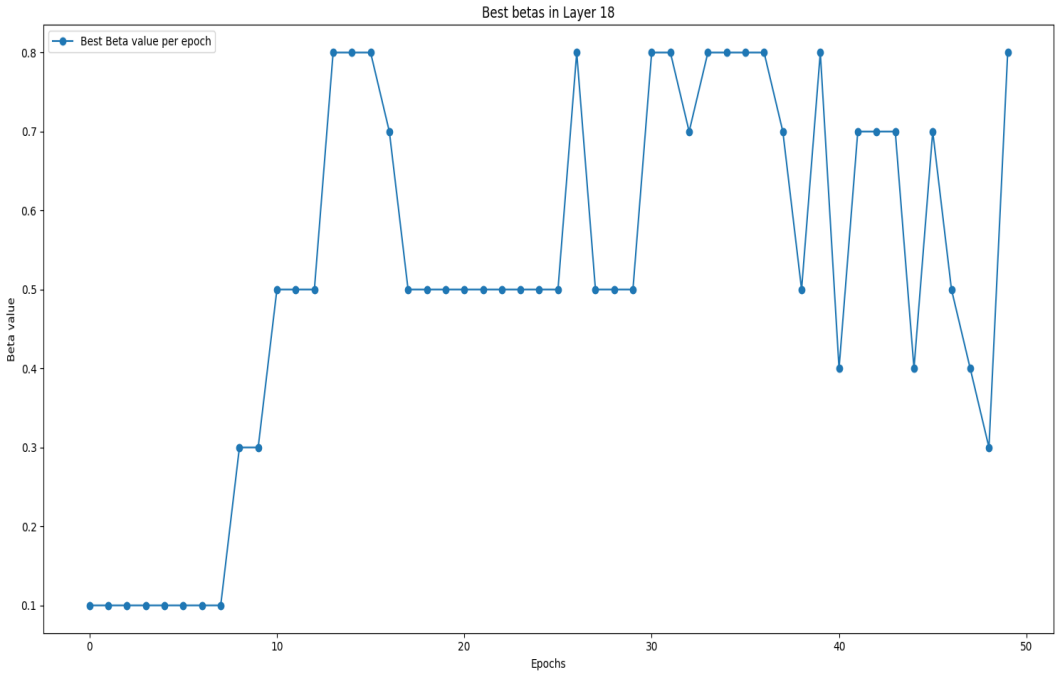


Figure 5.1: Epoch-wise "Best" momentum (β) values for the 18^{th} layer of MLP trained on FashionMNIST.

Next, we compare the ESM values over epochs for two representative hidden layers—one from the early part (Layer 2) and one from the deeper part (Layer 19) of the MLP—under two different momentum strategies: the default Batch Normalization with fixed $\beta = 0.1$, and the "Best β" configuration described earlier. This comparison is visualized in Figure 5.2.

The figure clearly demonstrates that across training epochs, the ESM values associated with the Best β strategy are consistently lower than those from the default momentum setting. Also the magnitude of estimation shift across layers is observed to be lower in Best β strategy. This suggests that adapting the momentum parameter can help reduce estimation shift during training.

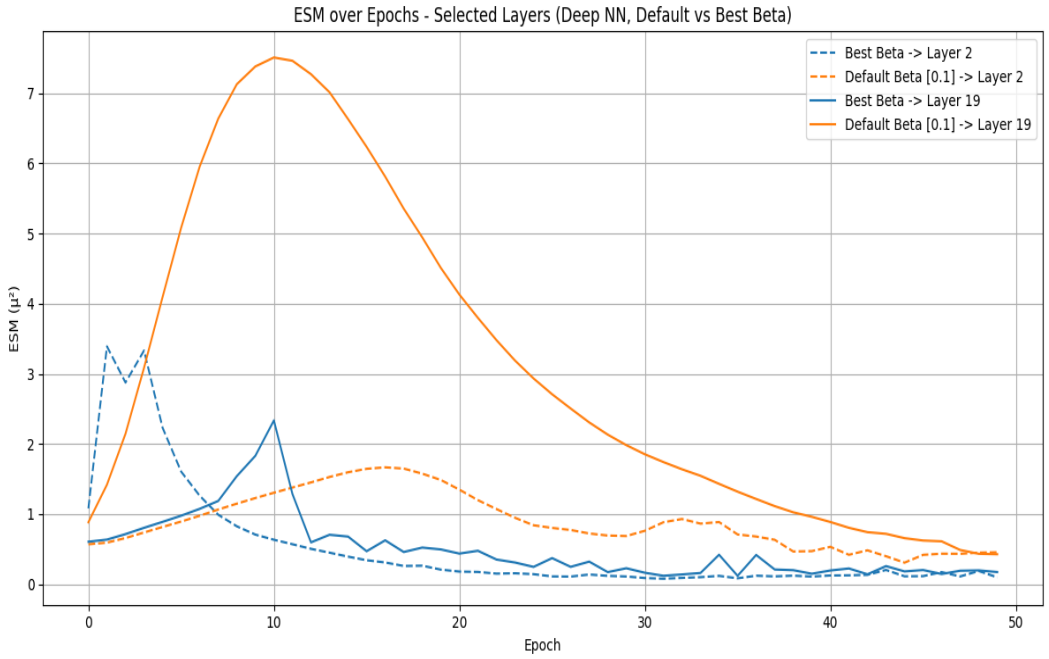


Figure 5.2: Comparison of Expected Squared Mean (ESM) across epochs for two selected layers (Layer 2 and layer 19) in MLP trained on FashionMNIST. Both Best β and Default $\beta = 0.1$ are shown.

We further investigate how our Best β strategy compares to the BN-BFN alternating normalization method, which was proposed to address estimation shift by switching between Batch Normalization and Batch Free Normalization across layers. Figure 5.3 shows the ESM values across training epochs for two representative layers in the MLP trained on the FashionMNIST dataset, evaluated under Best β and BN-BFN settings.

While BN-BFN achieves improvement over the default fixed-momentum configuration—particularly by lowering ESM in deeper layers—our results demonstrate that the Best β configuration consistently achieves further reduction in ESM across training.

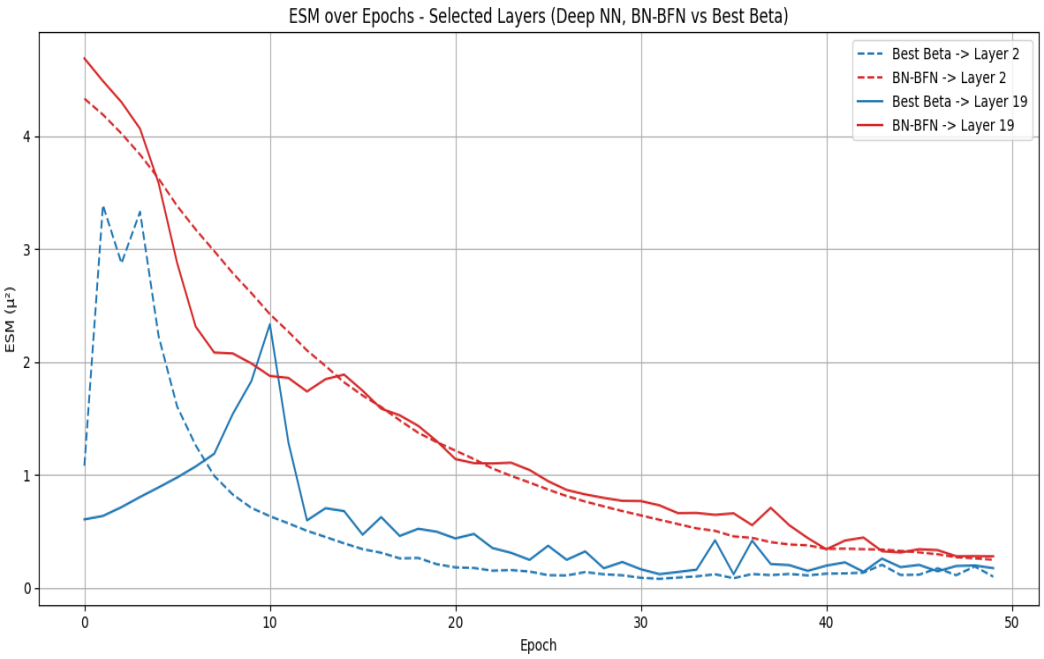


Figure 5.3: Comparison of Expected Squared Mean (ESM) across epochs for two selected layers in MLP (Layer 2 and layer 19) in MLP trained on FashionMNIST, comparing Best β strategy and BN-BFN method.

MNIST Results (MLP)

Evaluation on MNIST Dataset

We begin our quantitative evaluation by comparing various Batch Normalization (BN) strategies on the MNIST dataset using 20 layer MLP architecture. Table 5.1 presents the accuracy, sensitivity, specificity, and F1-score for each variant under two training regimes: full-batch and mini-batch (batch size = 100).

Full-Batch Training. In the full-batch setting, the default BatchNorm configuration using a fixed momentum $\beta = 0.1$ achieved an accuracy of 98.3%, sensitivity of 98.3%, specificity of 99.8% and F1-score of 98.3%, which progressively improved with higher momentum values until $\beta = 0.6$. The BN-BFN alternating method achieved an accuracy of 99.2%, sensitivity of 99.2%, specificity of 99.9% and F1-score of 99.2%.

The best performance was obtained by the *Dynamic Momentum BN (DMBN)* configuration, which adaptively selects β at each epoch and layer. This variant achieved an accuracy of 99.6%, sensitivity of 99.6%, specificity of 100% and F1-score of 99.6%, clearly outperforming the metrics obtained in static momentum settings as well as BN-BFN alternating scheme.

Mini-Batch Training. When trained using a batch-size of 100, overall performance

dropped marginally across all models, which is expected due to the inherent noise introduced by mini-batch gradient updates. The default BN configuration achieved an accuracy of 97.7%, sensitivity of 97.7%, specificity of 99.7% and F1-score of 97.7%. The BN-BFN alternating method achieved an accuracy of 97.8%, sensitivity of 97.8%, specificity of 99.8% and F1-score of 97.8%.

DMBN configuration achieved an accuracy of 98.0%, sensitivity of 98.0%, specificity of 99.8% and F1-score of 98.0% thereby demonstrating a superior performance.

Table 5.1: Evaluation metrics on MNIST using a 20-layer MLP with full batch and mini-batch training

Dataset	Model Variant	Accuracy	Sensitivity	Specificity	F1-Score
MNIST (Full Batch)	Static BN ($\beta = 0.1$) (default)	0.983	0.983	0.998	0.983
	Static BN ($\beta = 0.2$)	0.992	0.992	0.999	0.992
	Static BN ($\beta = 0.3$)	0.993	0.993	0.999	0.993
	Static BN ($\beta = 0.4$)	0.993	0.993	0.999	0.993
	Static BN ($\beta = 0.5$)	0.993	0.993	0.999	0.993
	Static BN ($\beta = 0.6$)	0.995	0.995	0.999	0.995
	Static BN ($\beta = 0.7$)	0.991	0.991	0.999	0.991
	Static BN ($\beta = 0.8$)	0.994	0.994	0.999	0.994
	Static BN ($\beta = 0.9$)	0.994	0.994	0.999	0.994
	DMBN (Best β)	**0.996**	**0.996**	**1.000**	**0.996**
	BN-BFN Alternating	0.992	0.992	0.999	0.992
MNIST (Batch size = 100)	Static BN ($\beta = 0.1$) (default)	0.977	0.977	0.997	0.977
	Static BN ($\beta = 0.2$)	0.976	0.975	0.997	0.975
	Static BN ($\beta = 0.3$)	0.974	0.974	0.997	0.974
	Static BN ($\beta = 0.4$)	0.976	0.975	0.997	0.975
	Static BN ($\beta = 0.5$)	0.976	0.976	0.997	0.976
	Static BN ($\beta = 0.6$)	0.975	0.974	0.997	0.974
	Static BN ($\beta = 0.7$)	0.972	0.972	0.997	0.972
	Static BN ($\beta = 0.8$)	0.975	0.975	0.997	0.975
	Static BN ($\beta = 0.9$)	0.973	0.972	0.997	0.972
	DMBN (Best β)	**0.980**	**0.980**	**0.998**	**0.980**
	BN-BFN Alternating	0.978	0.978	0.998	0.978

FashionMNIST Results (MLP)

Evaluation on FashionMNIST Dataset

Next we evaluated different Batch Normalization strategies on FashionMNIST, reporting accuracy, sensitivity, specificity, and F1-score for both full-batch and mini-batch (batch size = 100) setups (Table 5.2).

Full-Batch Training. The default model ($\beta = 0.1$) performed modestly achieving an accuracy of 80.2%, sensitivity of 80.2%, specificity of 97.8% and F1-score of 78.6%. The BN-BFN alternating method achieved an accuracy of 85.3%, sensitivity of 85.3%, specificity of 98.4% and F1-score of 85.5%.

DMBN configuration achieved an accuracy of 88.9%, sensitivity of 88.9%, specificity of 98.8% and F1-score of 88.8% thereby demonstrating a superior performance.

Mini-Batch Training. The default model ($\beta = 0.1$) achieved an accuracy of 87.3%, sensitivity of 87.3%, specificity of 98.6% and F1-score of 87.4%. The BN-BFN alternating method achieved an accuracy of 66.2%, sensitivity of 62.2%, specificity of 95.8% and F1-score of 58.3% likely due to overfitting.

DMBN configuration achieved an accuracy of 87.5%, sensitivity of 87.5%, specificity of 98.6% and F1-score of 87.4%. thereby demonstrating a superior performance.

CIFAR-10 Results (ResNet-50)

Evaluation on CIFAR-10 Dataset

The CIFAR-10 dataset was used to evaluate various normalization strategies within a deeper architecture,ResNet-50, under mini-batch training (batch size = 100), and the results are presented in Table 5.3.

The default model ($\beta = 0.1$) achieved an accuracy of 88.3%, sensitivity of 88.5%, specificity of 98.7% and F1-score of 88.3%. The BN-BFN alternating method achieved an accuracy of 88.4%, sensitivity of 88.4%, specificity of 98.7% and F1-score of 88.4%.

DMBN configuration achieved an accuracy of 88.7%, sensitivity of 89.0%, specificity of 98.7% and F1-score of 88.7%, demonstrating a superior performance over BN-BFN alternating scenario. However, when $\beta = 0.7$ was used as a static parameter, it achieved 89.7% accuracy, 89.8% sensitivity, 98.9% specificity, and 89.7% F1-score.

Table 5.2: Evaluation metrics on FashionMNIST using a 20-layer MLP with full batch and mini-batch training

Dataset	Model Variant	Accuracy	Sensitivity	Specificity	F1-Score
FashionMNIST (Full Batch)	Static BN ($\beta = 0.1$) (default)	0.802	0.802	0.978	0.786
	Static BN ($\beta = 0.2$)	0.858	0.858	0.984	0.855
	Static BN ($\beta = 0.3$)	0.871	0.871	0.986	0.868
	Static BN ($\beta = 0.4$)	0.866	0.866	0.985	0.864
	Static BN ($\beta = 0.5$)	0.864	0.864	0.985	0.866
	Static BN ($\beta = 0.6$)	0.859	0.859	0.984	0.858
	Static BN ($\beta = 0.7$)	0.769	0.769	0.974	0.763
	Static BN ($\beta = 0.8$)	0.880	0.880	0.987	0.877
	Static BN ($\beta = 0.9$)	0.836	0.836	0.982	0.835
	DMBN (Best β)	**0.889**	**0.889**	**0.988**	**0.888**
	BN-BFN Alternating	0.853	0.853	0.984	0.855
FashionMNIST (Batch size = 100)	Static BN ($\beta = 0.1$) (default)	0.873	0.873	0.986	0.874
	Static BN ($\beta = 0.2$)	0.866	0.866	0.985	0.865
	Static BN ($\beta = 0.3$)	0.872	0.872	0.986	0.871
	Static BN ($\beta = 0.4$)	0.872	0.872	0.986	0.872
	Static BN ($\beta = 0.5$)	0.872	0.872	0.986	0.872
	Static BN ($\beta = 0.6$)	0.857	0.857	0.984	0.857
	Static BN ($\beta = 0.7$)	0.873	0.873	0.986	0.873
	Static BN ($\beta = 0.8$)	0.863	0.863	0.985	0.862
	Static BN ($\beta = 0.9$)	0.864	0.864	0.985	0.865
	DMBN (Best β)	**0.875**	**0.875**	**0.986**	**0.874**
	BN-BFN Alternating	0.622	0.622	0.958	0.583

Table 5.3: Evaluation metrics on CIFAR-10 using ResNet-50 with mini-batch training

Dataset	Model Variant	Accuracy	Sensitivity	Specificity	F1-Score
CIFAR-10 (Batch size = 100)	Static BN ($\beta = 0.1$) (default)	0.883	0.885	0.987	0.883
	Static BN ($\beta = 0.2$)	0.894	0.895	0.988	0.894
	Static BN ($\beta = 0.3$)	**0.897**	**0.898**	**0.989**	**0.897**
	Static BN ($\beta = 0.4$)	0.831	0.837	0.981	0.832
	Static BN ($\beta = 0.5$)	0.833	0.835	0.981	0.833
	Static BN ($\beta = 0.6$)	0.877	0.878	0.986	0.877
	Static BN ($\beta = 0.7$)	0.884	0.886	0.987	0.884
	Static BN ($\beta = 0.8$)	0.865	0.867	0.985	0.865
	Static BN ($\beta = 0.9$)	0.881	0.884	0.987	0.881
	DMBN (Best β)	0.887	0.890	0.987	0.887
	BN-BFN Alternating	0.884	0.884	0.987	0.884

CIFAR-100 Results (ResNet-50)

Table 5.4: Evaluation metrics on CIFAR-100 using ResNet-50 with mini-batch training

Dataset	Model Variant	Accuracy	Sensitivity	Specificity	F1-Score
CIFAR-100 (Batch size = 100)	Static BN ($\beta = 0.1$) (default)	0.833	0.833	0.998	0.833
	Static BN ($\beta = 0.2$)	0.840	0.840	0.998	0.841
	Static BN ($\beta = 0.3$)	0.832	0.832	0.998	0.833
	Static BN ($\beta = 0.4$)	0.797	0.797	0.998	0.797
	Static BN ($\beta = 0.5$)	0.828	0.828	0.998	0.828
	Static BN ($\beta = 0.6$)	0.693	0.693	0.997	0.693
	Static BN ($\beta = 0.7$)	**0.857**	**0.857**	**0.999**	**0.857**
	Static BN ($\beta = 0.8$)	0.843	0.843	0.998	0.843
	Static BN ($\beta = 0.9$)	0.811	0.811	0.998	0.811
	DMBN (Best β)	0.848	0.848	0.998	0.848
	BN-BFN Alternating	0.846	0.846	0.998	0.846

Evaluation on CIFAR-100 Dataset

Table 5.4 presents the performance of various Batch Normalization strategies on the CIFAR-100 dataset using ResNet-50 trained with mini-batches of size 100.

The default model ($\beta = 0.1$) achieved an accuracy of 83.3%, sensitivity of

83.3%, specificity of 99.8% and F1-score of 83.3%. The BN-BFN alternating method achieved an accuracy of 84.6%, sensitivity of 84.6%, specificity of 99.8% and F1-score of 84.6%.

DMBN configuration achieved an accuracy of 84.8%, sensitivity of 84.8%, specificity of 99.8% and F1-score of 84.8%, demonstrating a superior performance over BN-BFN alternating scenario. However, when $\beta = 0.7$ was used as a static parameter, it achieved 89.7% accuracy, 85.7% sensitivity, 99.9% specificity, and 85.7% F1-score.

DMBN consistently outperformed both default β and BN-BFN alternating strategies across various datasets and model configurations. However, for CIFAR-10 and CIFAR-100, a particular beta ($\beta = 0.3$ for CIFAR-10 and $\beta = 0.7$ for CIFAR-100) when used as static parameter obtained better performance metrics than the DMBN. It may be possible that DMBN introduces noise in inner layers in different epochs which may not be a case in static β paramemter.

Chapter 6

Conclusion and Future Work

This dissertation explored the role of the momentum hyperparameter (β) in Batch Normalization (BN) and studied the impact of dynamically changing it in deep neural networks. Through a series of experiments on both simple and complex architectures—ranging from a Multi-Layer Perceptron (MLP) trained on MNIST and FashionMNIST to a ResNet-50 trained on CIFAR-10 and CIFAR-100—we systematically evaluated the impact of static, dynamic, and hybrid momentum strategies on model performance.

The results demonstrated that a fixed momentum value, while commonly used in practice, is suboptimal for many layers and stages of training. Our experiments showed that the optimal momentum varies both across layers and over epochs. This was clearly illustrated by the epoch-wise "Best β" plot, which highlighted non-monotonic and dynamic behavior in the preferred momentum values. In response to this observation, we introduced and evaluated an alternative: a **Dynamic Momentum BN** strategy that adjusts β adaptively based on **best** β configuration i.e. it applies the epoch-wise optimal values per layer, computed via Estimation Shift Magnitude (ESM) minimization. Additionally, we studied the **BN-BFN Alternating** approach that alternates between standard Batch Normalization and Batch Free Normalization [2].

Empirical results across datasets and training configurations indicated that:

- The proposed method, DMBN, achieved higher accuracy, sensitivity, specificity and F1-score values while comparing with the default β for MNIST, FashionMNIST, CIFAR-10 and CIFAR-100 datasets.

- DMBN, in comparison to BN-BFN, also achieved higher accuracy, sensitivity, specificity and F1-score in MNIST, FashionMNIST, CIFAR-10 and CIFAR-100 datasets.

In conclusion introducing dynamic momentum in Batch Normalization improve the model performance across various evaluation metrics.

Future Work

While the current study focused on performance metrics such as accuracy, sensitivity, specificity, and F1-score, further research can delve into the theoretical underpinnings of ESM-based momentum adaptation and its relationship with training stability and generalization. Moreover, extending the best β scheduling idea into a learnable, differentiable form during training (e.g. a formula based on various parameters like batch-size, activation function etc..) could make adaptive momentum fully autonomous and scalable.

Another promising direction is to integrate DMBN in multiscale deep learning architectures to validate its utility in various fields such as biomedical image segmentation, object detection, and object classification.

Bibliography

[1] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[2] Lei Huang et al. *Delving into the Estimation Shift of Batch Normalization in a Network*. 2022. arXiv: 2203.10778 [cs.CV].

[3] Yuxin Wu and Kaiming He. "Group Normalization". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 3–19. arXiv: 1803.08494 [cs.CV].

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].

[5] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. "Instance Normalization: The Missing Ingredient for Fast Stylization". In: *arXiv preprint arXiv:1607.08022*. 2016. arXiv: 1607.08022 [cs.CV].

[6] Shibani Santurkar et al. *How Does Batch Normalization Help Optimization?* 2019. arXiv: 1805.11604 [stat.ML].

[7] Sergey Ioffe. *Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models*. 2017. arXiv: 1702.03275 [cs.LG].

[8] Hongwei Yong et al. "Momentum Batch Normalization for Deep Learning with Small Batch Size". In: Glasgow, United Kingdom: Springer-Verlag, 2020, pp. 224–240. ISBN: 978-3-030-58609-6. DOI: 10.1007/978-3-030-58610-2_14.

[9] URL: https://pytorch.org/docs/stable/generated/torch.nn.Linear.html.

[10] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[11] URL: https://en.wikipedia.org/wiki/MNIST_database#:~:text=The%20MNIST%20database%20(Modified%20National,the%20field%20of%20machine%20learning.

[12] URL: https://www.kaggle.com/datasets/zalando-research/fashionmnist.

[13] URL: https://www.cs.toronto.edu/~kriz/cifar.html.

[14] URL: https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html.

[15] URL: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html.