This ICCV workshop paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

# Leveraging Batch Normalization for Vision Transformers

Zhuliang Yao[1,2*] Yue Cao[2] Yutong Lin[2,3*] Ze Liu[2,4*] Zheng Zhang[2] Han Hu[2]
[1] Tsinghua University
[2] Microsoft Research Asia
[3] Xi'an Jiaotong University
[4] University of Science and Technology of China

## Abstract

*Transformer-based vision architectures have attracted great attention because of the strong performance over the convolutional neural networks (CNNs). Inherited from the NLP tasks, the architectures take Layer Normalization (LN) as a default normalization technique. On the other side, previous vision models, i.e., CNNs, treat Batch Normalization (BN) as a de facto standard, with the merits of faster inference than other normalization layers due to an avoidance of calculating the mean and variance statistics during inference, as well as better regularization effects during training.*

*In this paper, we aim to introduce Batch Normalization to Transformer-based vision architectures. Our initial exploration reveals frequent crashes in model training when directly replacing all LN layers with BN, contributing to the un-normalized feed forward network (FFN) blocks. We therefore propose to add a BN layer in-between the two linear layers in the FFN block where stabilized training statistics are observed, resulting in a pure BN-based architecture. Our experiments proved that our resulting approach is as effective as the LN-based counterpart and is about 20% faster.*

## 1. Introduction

Transformer [30] is initially proposed for Natural Language Processing (NLP) tasks. The great success in this field encourages the researchers in computer vision community to apply Transformer in vision tasks. Along this direction, Vision Transformer (ViT) [8] and Data-Efficient Image Transformers (DeiT) [27] are two pioneers to make Transformer based architecture work effective on an important task of image recognition. Swin Transformer [19] further introduces certain inductive biases, such as locality, hierarchy and translation invariance, into the design of Transformer architectures, resulting in a general-purpose backbone which achieves the state-of-the-art accuracy on various vision benchmarks, such as COCO object detection and ADE20K semantic segmentation.

While vision Transformer achieves strong performance, we note the de facto standard of CNNs, i.e., Batch Normalization (BN) [16], is not included or well-studied in it, probably because vision Transformers directly inheriting the LN layers for that of standard Transformers for NLP tasks. BN is proposed by Ioffe and Szegedy to make neural networks converge faster and more stable by re-centering and re-scaling the activations. The effectiveness is widely proved by the past success in vision tasks. However, early attempts of using BN in NLP tasks faced significant performance degradation [22]. On the other hand, Layer Normalization (LN) [1] seems born suitable for variable length input. So Transformer has incorporated LN instead of BN as their default normalization scheme.

Although both BN and LN normalizes the activation of each layer by mean and variance statistics, the different ways to compute statistics make the training dynamics different. Compared to LN, BN shows better robustness and generalization ability for vision tasks. Also, BN has an advantage that it is in generally faster in inference than other batch-unrelated normalizations such as LN, due to an avoidance of calculating the mean and variance statistics during inference. Further more, this offline statistic characteristic in inference provides a possibility of building Inception-like convolutional blocks [6, 7], which has noticeable performance gain and no extra time cost in inference.

Despite of the advantages of BN and the past experience of CNNs, most Transformer-based vision architectures just inherits LN from Transformer and pays rare attentions on BN. CvT and CeiT adopt BN after Convolution layers in proposed new modules, but keep all LN layers in other original modules. MoCo v3 [3] evaluates the performance of replacing LN with BN in FFN blocks of ViT to shrink the systematic gap between ViT and ResNets, but its replacement is incomplete and lack of further analysis.

In this paper, we aim to introduce BN layers into the

*This work is done when Zhuliang Yao, Yutong Lin, Ze Liu are interns at Microsoft Research Asia. Correspondence to: Yue Cao (yuecao@microsoft.com).

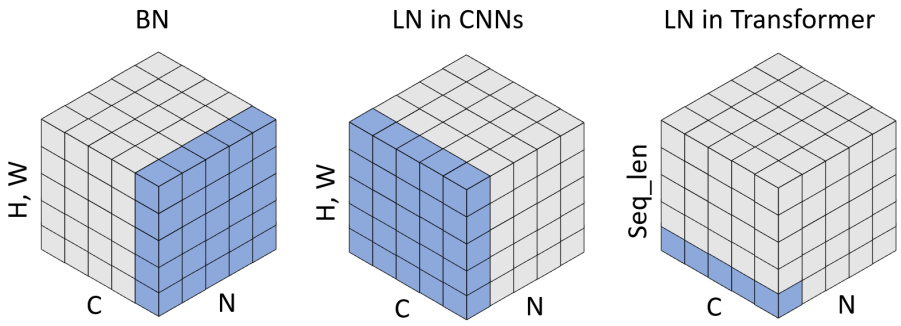


Figure 1: An illustration of **normalization methods**. Each subplot represents a feature map tensor, with B as the batch axis, C as the channel axis, and (H, W) or Seq_len as the spatial axes. The elements in blue are normalized by the same mean and variance, computed by aggregating the values of these elements.

vision Transformers. Our direct application of using BN instead of LN results in frequent crashes in model training. To investigate this phenomenon, we proposed eXtended Signal Propagation Plots (XSPPs) from SPPs [2] to adapt the difference of statistics calculation between BN and LN. Then we monitor the XSSPs during training and find that most of such crashes are due to the un-normalized FFN blocks. We thus propose to add a BN layer in-between the two linear layers in each FFN block. The effectiveness of this simple modification is proved not only by observed stabilized training XSPPs but also the on-par results with LN-based ViT and Swin-T backbones on practical vision tasks. Besides, ascribed to the use of BN, our vision transformers easily acquire 20% speed performance gain without any special optimizations.

## 2. Related Work

**Transformer-based vision backbones.** Vision Transformer (ViT) [8] is the pioneering work to show the potential of transformer based backbones for vision tasks. With large scale training (ImageNet-21k, JFT-300M), it achieves impressive results, which are comparable with large ResNets [11]/EfficientNets [26]. Its follow up, DeiT [27], adopts several training strategies to make it efficient with ImageNet-1k training. Besides, PVT [32] and Swin Transformer [19] both introduce a hierarchical structure into vision transformer. This design makes them suitable for downstream dense prediction tasks and Swin Transformer with further locality inductive bias achieves state-of-the-art results on COCO detection task and ADE semantic segmentation task.

There are also some follow-ups introducing convolutional modules to transformer architectures. CvT [34] introduces convolutional token embedding layers and extends the attention module with convolutional projection layers. CeiT [39] also proposes a convolutional Image-To-Tokens module. In addition, it inserts convolution layers in-between feed forward networks. Visual Transformer [33] utilizes several convolutional layers to extract low-level features and then tokenize these features.

**Convolution based vision backbones.** Convolution based backbones have become mainstream since the propose of AlexNet [18], which shows triumphant performance at the ILSVRC 2012 contest [21] and changes the landscape of deep learning. Since then, a lot of convolution based networks have been proposed, like VGG [23], Inception [24], ResNet [11], DenseNet [13], HRNet [31] and EfficientNet [26]. Most of the recent convolution based architectures adopt BN as the default normalization layer and show that BN is in general faster in inference and achieves better performance than other batch-unrelated normalization.

**Normalization methods.** It is well-known that normalizing the feature maps makes the training faster and more stable [35]. To normalize the activations, a variety of normalization methods like BN [16], LN [1], IN [29], GN [35] are developed for general tasks or specific applications.

All these normalization methods can be categorized into batch-related methods and batch-irrelevant methods. Batch-related methods include BN and its variants like SyncBN [20], BRN [15], CBN [38] and so on. They all treat the batch data as a whole. The batch dimension is used in the calculations of both mean and variance. To overcome the problems of batch size changes or nonexistence, many new data dimensions are explored for providing more statistic data. Another solution is exclude the batch concept from statistics, leading to batch-irrelevant methods. Among all these methods, LN [1] shows the best performance on NLP tasks, GN [35] is the most powerful competitor to BN which achieved good performance especially for dense prediction

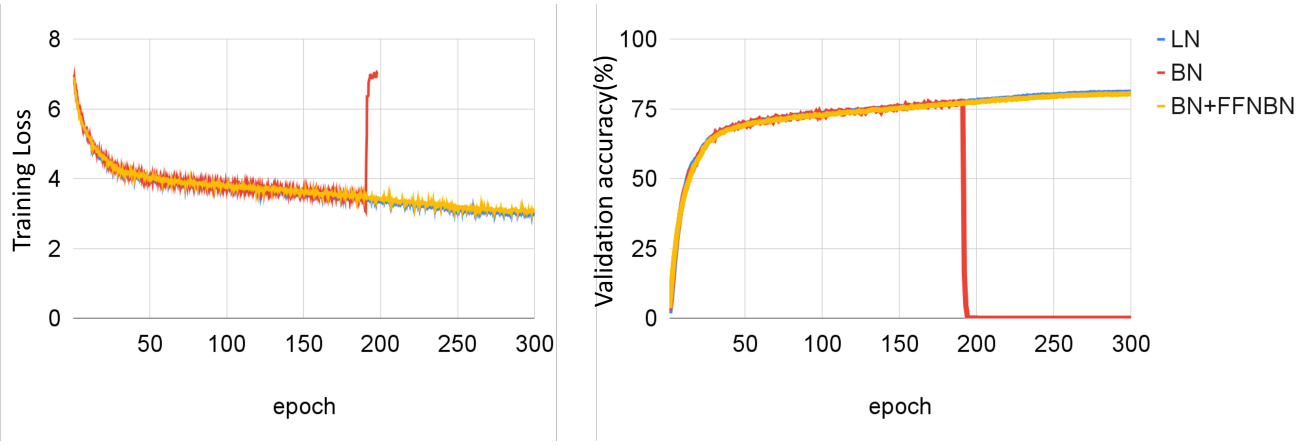


Figure 2: The training loss and validation accuracy of Swin-T with LN, BN, BN+FFNBN on ImageNet-1K.

tasks, such as object detection, and IN [29] owns the best artistic ability.

## 3. Method

We describe the exploration of pure BN-based vision transformer design space in this section. First, we discuss the relationships of various normalization techniques. Then we show the different normalization choices for different neural networks, and point out that Transformer-based vision architectures meets the need of Batch Normalization well. Finally, we use our proposed eXtended Signal Propagation Plots to build a robust pure BN-based Swin Transformer, as well as show a fusion technique for speedup.

### 3.1. BN v.s. LN

As one of the most important components in deep neural networks, normalization technique has achieved much attention in literature.

Among all those methods, Batch Normalization (BN) [16] plays the most important role for vision tasks in convolutional neural networks (CNNs). It mainly normalizes the input by re-centering and re-scaling, which shows great regularization ability. Given a 4D tensor input $(N, C, H, W)$, BN has $2C$ elements of statistics, where each mean and variance value are computed across $(N, H, W)$ in training while running mean and variance are used in inference. This design makes BN much faster and even possible to be fused into other layers in inference [6].

However, the effect of BN is dependent on the batch-size and it is not obvious how to apply BN to recurrent neural networks. Therefore Layer Normalization [1] as well as some other batch-irrelevant normalization methods are designed to acquire statistics for each sample independently. In detail, LN normalizes the input along $(C)$ leading to $2NHW$ statistical values. But this choice requires statistic calculations in both training and inference, which is noticeably slower.

### 3.2. Normalizations for Different Architectures

Contributed to the diversity of normalization methods above, all kinds of neural networks can always select the one that suits their applied tasks the most.

For CNNs, BN became the most widely used normalization layer soon after it was proposed in 2015. Most breakthroughs of CNNs, such as Inception V2/V3 [24], ResNet [11], DenseNet [13], MobileNet [12], EfficientNet [26] choose BN as the default normalization method. Although there do exist some unique normalization methods for a particular group of scenarios, such as GN for small batch-size regime [35] and IN for style transfer [9], they could hardly match the comprehensive performance of BN or provide similar regularization ability like BN.

For Transformer, its development is closely coupled with LN. The main reason is that almost all NLP tasks take variable length sequences as input, which is very suitable for LN that only calculates statistics in the channel dimension without involving the batch and sequence length dimension. Note that the concept of LN in Transformer is different from the one in CNNs, as Figure 1 illustrated. LN in Transformer has exactly the same formulation with Channel Normalization (CN) [4] in CNNs.

For Transformer-based vision architectures, a lot of design principles are inherited from Transformer in NLP, including LN as the default normalization method. But the input is switched from sequences with variable length to usual fixed size images, which means the stable and sufficient data for statistics offers new possibilities for applying BN in these architectures.

### 3.3. Pure BN-based Vision Transformers

**Initial attempt with convergence problem** As an initial attempt, we conduct a straightforward experiment that all LN layers are directly replaced by BN layers. Our model base

on the tiny version of the standard Swin Transformer (Swin-T) [19], and all the other hyperparameters follow Swin-T's official settings. Unexpectedly, this plain design leads to convergence problem, i.e., the model is very unstable to frequently crash during training. The training curve and validation accuracy in Figure 2 with BN-try{1,2,3} reveal that the crash happens suddenly and irregularly. We hypothesize this crash is originated from exceeding the current local optima, which may be observed with some abnormal statistics in the hidden activations.

**Analysis with eXtended Signal Propagation Plots** Generally, the statistics of the hidden activations are believed to be extremely beneficial for finding out the key reason that contributing to frequent crashes. We notice that Signal Propagation Plots (SPPs) [2] is introduced for similar purpose as a simple set of visualizations which help to inspect signal propagation in neural networks. Since SPPs is originally proposed for deep ResNets with BN, We modify some statistic calculations of original SPPs to meet our needs on vision transformer with both BN and LN, noted as eXtended Signal Propagation Plots (XSPPs). We use XSPPs not only for initialization but also the checkpoints during training.

Average Feature Squared Mean (AFSM) computes the average span along the corresponding normalization's feature axis, and then averages it on the left axis:

$$\begin{aligned} AFSM_{BN} =& \frac{1}{C}\sum_{c}\left(\frac{1}{N*H*W}\sum_{n,i,j}(F_{n,i,j,c})\right)^2 \\ AFSM_{LN} =& \frac{1}{N*H*W}\sum_{n,i,j}\left(\frac{1}{C}\sum_{c}(F_{n,i,j,c})\right)^2 \end{aligned} \tag{1}$$

Average Feature Variance (AFV) computes the variance across the corresponding normalization's feature axis, and then averages on the left axis:

$$\begin{aligned} AFV_{BN} =& \frac{1}{C}\sum_{c}\left(\frac{1}{N*H*W}\sum_{n,i,j}\left(\right.\right. \\ & \left.\left. F_{n,i,j,c} - \frac{1}{N*H*W}\sum_{n,i,j}(F_{n,i,j,c})\right)^2\right) \\ AFV_{LN} =& \frac{1}{N*H*W}\sum_{n,i,j}\left(\right. \\ & \left.\frac{1}{C}\sum_{c}\left(F_{n,i,j,c} - \frac{1}{C}\sum_{c}(F_{n,i,j,c})\right)^2\right) \end{aligned} \tag{2}$$

Average Feature Variance on Residual (AFVR) computes similar statistics as AFV, but uses the hidden activation at the end of each residual branch. It could be considered as the differential value of AFVR.

Though this tool only takes the forward pass into consideration, many previous analyses [25, 37, 10, 2] have pointed out that for the backward pass of networks with shortcuts will typically neither explode nor vanish so long as the signal on the forward pass is well behaved. Besides, we also observe that the spike always appear in hidden activations of forward first and then in the following gradient of backward.

With XSPPs, we observe several meaningful patterns as plotted in Figure 3. Here the y-axis denotes the values of corresponding statistics. And the x-axis denote the index of the residual block. For Swin-T, there are (2, 2, 6, 2) encoder blocks for each stage, while each encoder block contains two residual blocks, i.e., the attention block and the Feed-Forward-Network (FFN) block. The trained model of BN is the checkpoint of last epoch before crash. The trained model of LN is the checkpoint at the end of training, and we also evaluate the checkpoint at same epoch of BN's model with no trend effect of these statistics.

First, both initialized models of LN and BN have controllable statistics, which is consistent with our observation that both models will not face crash problems at beginning. Second, all statistics of AFSM and AFV positively correlate to the block depth inside each stage, contributing to the positive increment from every residual branches. Third, BN model has much larger AFSM statistic than that of LN model, especially in Stage 1 and Stage 3.[1] Because Stage 1 has the largest feature dimension and Stage 3 has the most residual blocks. Lastly, the AFVR at even layers are always larger than the one at odd layers for both LN and BN. This indicates that FFN blocks contributes a lot more to the rapid growth than attention blocks.

**Solution to stable pure BN-based vision transformer** Clearly, the convergence problem is mainly contributed to the increment from the FFN blocks. We therefore propose to add a BN layer in-between the two linear layers in each FFN block, noted as BN+FFNBN. Then we investigate the dynamic changes of this model with our XSPPs. Although the initialized BN+FFNBN model (yellow line in Figure 3) seems slightly higher statistics than both BN and LN models, it is still acceptable and probably owing to more normalizations make the hidden activations closer to a variance of 1. Then we try to train such a BN+FFNBN model with same hyperparameters as above. The whole training procedure

[1]The AFV and AFVR statistics cannot be fairly compared between LN and BN, due to the intrinsic differences between channel dimension and batch & feature dimension, such as channel dimension contains less elements than batch & feature dimension and batch & feature dimension has locality due to downsample and shifted-window.

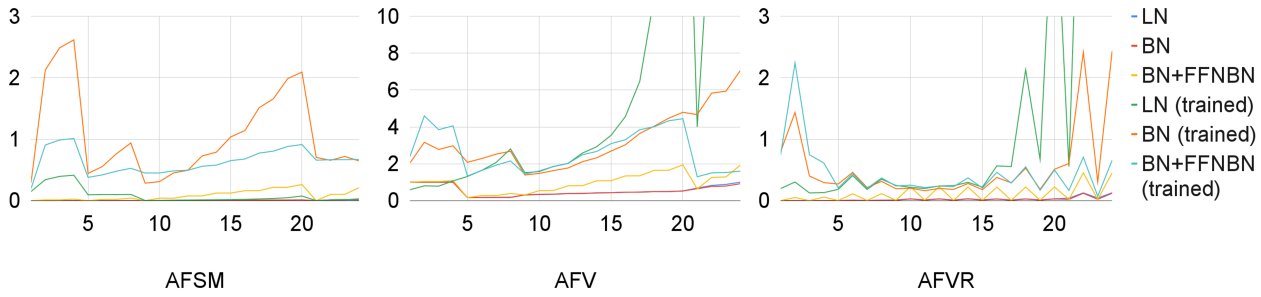


Figure 3: Signal Propagation Plot for a Swin-T tiny at initialization as well as trained.

is quite stable and we plot the trained BN+FFNBN model as the light blue line in Figure 3), where most abnormal trends are controlled. Compared to BN, BN+FFNBN model has more consistent AFSM and AFV over all four stages. Its AFSMs in Stage 1 and Stage 3 stay under a reasonable limitation like only 1 or 2. And the AFVR is significantly smaller than BN in Stage 4. Please also see Section 4 for performance comparison of image classification on ImageNet 1K [21].

### 3.4. Merging BN into Linear Layer

In CNNs, fusing BN and preceding convolution in inference is a quite mature technique for extra speedup. The scenario changes a little when pre-normalization is adopted as in vision transformers, that we need to merge the freezed BN layer to a subsequent linear ($1\times1$ conv) layer.

Given a feature map $F$ with the shape $(C, H, W)$, and the freezed BN layer's parameters $\hat{\mu}, \hat{\sigma}^2, \gamma, \beta \in \mathbb{R}^C$, the calculation of BN layer can be formulated as a $1\times1$ convolution. The $F$'s normalized version, $\widehat{F}$ at position $(i, j)$ is formulated as:

$$\begin{pmatrix} \widehat{F}_{1,i,j} \\ \widehat{F}_{2,i,j} \\ \vdots \\ \widehat{F}_{C-1,i,j} \\ \widehat{F}_{C,i,j} \end{pmatrix} = W \cdot \begin{pmatrix} F_{1,i,j} \\ F_{2,i,j} \\ \vdots \\ F_{C-1,i,j} \\ F_{C,i,j} \end{pmatrix} + \begin{pmatrix} \beta_1 - \gamma_1 \frac{\hat{\mu}_1}{\sqrt{\hat{\sigma}_1^2+\varepsilon}} \\ \beta_2 - \gamma_2 \frac{\hat{\mu}_2}{\sqrt{\hat{\sigma}_2^2+\varepsilon}} \\ \vdots \\ \beta_{C-1} - \gamma_{C-1} \frac{\hat{\mu}_{C-1}}{\sqrt{\hat{\sigma}_{C-1}^2+\varepsilon}} \\ \beta_C - \gamma_C \frac{\hat{\mu}_C}{\sqrt{\hat{\sigma}_C^2+\varepsilon}} \end{pmatrix},$$

$$W = \begin{pmatrix} \frac{\gamma_1}{\sqrt{\hat{\sigma}_1^2+\varepsilon}} & 0 & \cdots & & 0 \\ 0 & \frac{\gamma_2}{\sqrt{\hat{\sigma}_2^2+\varepsilon}} & & & \\ \vdots & & \ddots & & \vdots \\ & & & \frac{\gamma_{C-1}}{\sqrt{\hat{\sigma}_{C-1}^2+\varepsilon}} & 0 \\ 0 & & \cdots & 0 & \frac{\gamma_C}{\sqrt{\hat{\sigma}_C^2+\varepsilon}} \end{pmatrix}. \quad (3)$$

Thus the computation of BN and the linear layer can be expressed as

$$\widehat{f}_{i,j} = \boldsymbol{W}_{Linear} \cdot (\boldsymbol{W}_{BN} \cdot f_{i,j} + \boldsymbol{b}_{BN}) + \boldsymbol{b}_{Linear}, \quad (4)$$

where $f_{i,j} \in \mathbb{R}^C$ denotes the vectorized $F$ as position $(i, j)$, $\boldsymbol{W}_{BN} \in \mathbb{R}^{C\times C}, \boldsymbol{b}_{BN} \in \mathbb{R}^C$ denote the parameter of BN, and $\boldsymbol{W}_{Linear} \in \mathbb{R}^{C_{out}\times C}, \boldsymbol{b}_{Linear} \in \mathbb{R}^{C_{out}}$ denote the parameters of subsequent linear layer.

Obviously, these two layers can be replaced by one linear layer as:

$$\begin{aligned} \boldsymbol{W} &= \boldsymbol{W}_{Linear} \cdot \boldsymbol{W}_{BN} \\ \boldsymbol{b} &= \boldsymbol{W}_{Linear} \cdot \boldsymbol{b}_{BN} + \boldsymbol{b}_{Linear}. \end{aligned} \quad (5)$$

Thus the BN could still be merged to the linear layer in vision transformers for the speedup in inference.

## 4. Experiment

### 4.1. Supervised Classification

**Setup** For supervised classification, we strictly follow the setting in [19]. We adopt ImageNet-1K [5] as the benchmark dataset, which contains 1.28M training images and 50K validation images from 1,000 classes. The top-1 accuracy on a single crop is reported. We employ an AdamW [17] optimizer for 300 epochs using a cosine decay learning rate scheduler and 20 epochs of linear warm-up. A batch size of 1024, an initial learning rate of 0.001, and a weight decay of 0.05 are used. For all model variants, we adopt a default input image resolution of $224^2$. An increasing degree of stochastic depth augmentation is employed for larger models, i.e. $\{0.2, 0.3, 0.5\}$ for LN-based Swin&SwinD-{T, S, B}, $\{0.1, 0.2, 0.4\}$ for BN-based Swin&SwinD-{T, S, B}, respectively. DeiT-S adopts same setting as Swin-T. We train all these models on 8 NVIDIA V100 GPUs. Each experiment costs about 50, 100, 200 hours for T-, S-, B-level models.

**DeiT and Swin Transformer** We start by replacing the LN in DeiT and the original Swin to BN. Main results are

Table 1: Comparison of different backbones with different normalizations on ImageNet-1K classification.

| Architecture | Params | FLOPs | Normalization | Top-1 Acc (%) |
|---|---|---|---|---|
| DeiT-S | 22M | 4.6G | LN | 79.8 |
| | 22M | 4.6G | BN+FFNBN | 78.8 |
| Swin-T | 29M | 4.5G | LN | 81.2 |
| | 29M | 4.5G | BN+FFNBN | 80.9 |
| Swin-S | 50M | 8.7G | LN | 83.0 |
| | 50M | 8.7G | BN+FFNBN | 82.8 |
| Swin-B | 88M | 15.4G | LN | 83.3 |
| | 88M | 15.4G | BN+FFNBN | 83.1 |
| SwinD-T | 22M | 3.9G | LN | 81.1 |
| | 22M | 3.9G | BN+FFNBN | 81.4 |
| SwinD-S | 41M | 7.8G | LN | 82.2 |
| | 41M | 7.8G | BN+FFNBN | 82.4 |
| SwinD-B | 92M | 17.2G | LN | 83.1 |
| | 92M | 17.2G | BN+FFNBN | 83.1 |

Table 2: Analysis on different drop path rates for different architectures. Experiments are conducted with BN+FFNBN normalization. * indicates the best setting for LN.

| Architecture | Drop Path Rate | Top-1 Acc(%) |
|---|---|---|
| Swin-T | 0.05 | 80.7 |
| | 0.1 | **80.9** |
| | 0.2* | 80.6 |
| SwinD-T | 0.1 | **81.4** |
| | 0.2* | 81.0 |
| | 0.3 | 80.6 |
| SwinD-B | 0.3 | 82.0 |
| | 0.4 | **83.1** |
| | 0.5* | 82.5 |

shown in Table 1. Compared to the LN-based DeiT, BN-based DeiT suffers a significant drop (79.8% vs. 78.8%), this is probably because DeiT lacks some useful inductive bias which is inhibited by the good statistical properties of BN. This situation is mitigated for Swin as all BN-based models with different capacities show similar Top-1 accuracy with their LN-based counterpart. Considering the initial crash, this comparable result demonstrates the practicality of XSPPs and the reliability of the proposed BN+FFNBN method.

**Deeper Swin Transformers** Some recent works [28, 40] notice that vision transformers saturate fast and even fail during training when scaled to be deeper. However most CNNs performance can be always improved by stacking more layers. To verify if BN is one reason of CNN's surprising depth-expansion ability, we build a series of deeper Swin models called SwinD-{T, S, B}. Their network configurations are as follow:

- SwinD-T: C = 64, layer numbers = 2, 2, 18, 2
- SwinD-S: C = 64, layer numbers = 2, 2, 42, 2
- SwinD-B: C = 96, layer numbers = 2, 2, 42, 2

where $C$ denotes the channel number of the hidden layers in the first stage. Note we construct variants of different depths by varying only the layer number of the third stage, following [19, 11].

For all the experiment pairs of Swin and its corresponding SwinD with BN+FFNBN, the overall performance is comparable under similar parameters and FLOPs. One spotlight is that the 24 layers SwinD-T obtains 0.5% higher performance than Swin-T with even 24% less parameters and 13.3% less FLOPs, which may shed some light on efficient vision transformer design.

For the comparison between LN and BN+FFNBN under the same architectures, most BN+FFNBN models achieve slightly better results. Furthermore, our plain 48-layer models, i.e., SwinD-S and SwinD-B are trained smoothly without any crashes, which may proves our hypothesis that BN could help deeper networks work well.

### 4.2. Analysis

For analysis section, we use the same setting as described in Section 4.1. For the experiments in *Pre-Norm and Res-Post-Norm* and *Addition BN for all 1×1 convolution*, we use a 100-epoch cosine decay learning rate scheduler, and the model is specified as Swin-T. These two parts are conducted to deliver more observations and knowledge about the combination of BN and Transformer-based vision architectures.

Table 3: Analysis on Pre-Norm and Res-Post-Norm.

| Normalization | Pre-Norm | Res-Post-Norm | Top-1 Acc(%) |
|---|---|---|---|
| LN | | | 78.3 |
| | ✓ | | 78.3 |
| | | ✓ | 78.2 |
| | ✓ | ✓ | 79.1 |
| BN+FFNBN | | | 78.0 |
| | ✓ | | 78.0 |
| | | ✓ | 77.8 |
| | ✓ | ✓ | 78.4 |

Table 4: Analysis on additional BN for all 1×1 convolution layer.

| Normalization | Extra BN in attention | Extra BN in FFN | Top-1 Acc(%) |
|---|---|---|---|
| LN | | | 78.3 |
| | ✓ | | 78.4 |
| | ✓ | ✓ | 78.6 |
| BN | | | 78.0 |
| | ✓ | | 78.2 |
| | ✓ | ✓ | 78.4 |

**Drop path rate** Stochastic depth [14] has been proved as an effective regularization technique for Transformers. To achieve the best performance, we ablate the drop path rate for all the architectures. In Table 2, we list results of Swin-T, SwinD-T, SwinD-B with BN+FFNBN normalization. Appropriate drop path rates lead to 0.8% and 1.1% performance boost for SwinD-T and SwinD-B, respectively. Besides, we also observe that the optimal drop path rates for BN+FFNBN are consistently smaller than those for LN, indicating that BN may act as an implicit regularizer.

**Pre-Norm and Res-Post-Norm** In early versions of Transformer [30], LN is placed after the addition of residual branch and shortcut as called Post-Norm. Recent implementations adopt Pre-Norm in which LN is applied on the input of every sub-layer, which shows better performance than post-norm. However there is a design choice left as putting LN at the end of each residual block. To avoid confusion, we call it Res-Post-Norm. Ideally, Res-Post-Norm won't stop the gradient in main branch as Post-Norm did, and would better control the output of each residual block to avoid explosion than Pre-Norm.

The results are listed in Table 3. For both LN and BN+FFNBN, Pre-Norm keeps 0.1%-0.2% increase over Res-Post-Norm. Because both models have few troubles in convergence while Pre-Norm is indicated [36] as help gradients well-behaved at initialization. Besides, using Pre-Norm and Res-Post-Norm together leads to noticeable performance gain over using each one separately.

**Addition BN for all 1×1 convolution** BN is usually added after every convolutional layer in CNN practice. Here we propose two choices of adding extra BN in vision transformers, considering all linear layer as 1×1 convolution. One is adding BN after every linear layer in attention block except the query and key embedding layers. Another is adding BN after the two linear layer in FFN block. We show the results in Table 4. Obviously, more BN layers leads to better performance no matter what the Pre-Norm is. Adding as much BN as possible could bring 0.3%-0.4% Top-1 accuracy benefit.

### 4.3. Speedup Evaluation

**Layer-level benchmark for BN and LN** In order to show the original speed performance gap between BN and LN, we conduct a benchmark to compare the training and inference time of a single BN and LN layer in Table 5. This benchmark adopt the input shapes from the standard Swin-T model's 4 stages. The time cost in millisecond is calculated by feeding the batch data into one NVIDIA V100 GPU for 100,000 times. We use PyTorch as our experiment framework.

BN is generally faster than LN in early stages when input has larger spatial resolution. BN slows down along the channel number grows. Overall BN saves about 50% time cost of LN, because early stage contributes more to the total time cost.

**Model-level comparison of BN and LN** We run tests on Swin Transformer series to compare the end-to-end speed performance on model level. As Table 6 shows, BN+FFNBN is faster than LN in training but slower than LN in inference. The reasons are two folds. First, BN+FFNBN adds a extra

Table 5: **Layer-level benchmark for BN and LN** The time cost of BN and LN layers is counted in millisecond on a V100 GPU. The stages and input shapes are adopted from the standard Swin Transformer tiny model.

| Stage | 1 | | 2 | | 3 | | 4 | | Total |
|---|---|---|---|---|---|---|---|---|---|
| Input Shape [B,H,W,C] | [128,56,56,96] | | [128,28,28,192] | | [128,14,14,384] | | [128,7,7,768] | | - |
| # Blocks | single | 2× | single | 2× | single | 6× | single | 2× | - |
| LN | 14.07 | 28.13 | 3.92 | 7.84 | 1.19 | 7.12 | 0.39 | 0.77 | 43.87 |
| BN | 4.20 | 8.40 | 2.20 | 4.40 | 1.14 | 6.84 | 0.54 | 1.08 | 20.72 |
| Speedup | 70.14% | 70.14% | 43.88% | 43.88% | 3.93% | 3.93% | -39.66% | -39.66% | 52.77% |

(a) Training

| Stage | 1 | | 2 | | 3 | | 4 | | Total |
|---|---|---|---|---|---|---|---|---|---|
| Input Shape [B,H,W,C] | [128,56,56,96] | | [128,28,28,192] | | [128,14,14,384] | | [128,7,7,768] | | - |
| # Blocks | single | 2× | single | 2× | single | 6× | single | 2× | - |
| LN | 6.14 | 12.28 | 1.99 | 3.99 | 0.53 | 3.20 | 0.18 | 0.36 | 19.82 |
| BN | 1.83 | 3.67 | 0.81 | 1.61 | 0.41 | 2.47 | 0.26 | 0.52 | 8.27 |
| Speedup | 70.14% | 70.14% | 59.53% | 59.53% | 22.65% | 22.65% | -44.40% | -44.40% | 58.28% |

(b) Inference

Table 6: **Model-level comparison of BN and LN** Standard Swin Transformer models is used. Throughput is measured using a V100 GPU, following [27]. * denotes fusing BN into linear layers.

| Model | Normalization | Params | FLOPs | Training throughput (images/s) | Inference throughput (images/s) |
|---|---|---|---|---|---|
| Swin-T | LN | 28.29M | 4.49G | 270.6 | 755.2 |
| | BN+FFNBN | 28.32M | 4.50G | **291.2** | 675.9 |
| | BN+FFNBN* | 28.26M | 4.49G | - | **882.7** |
| Swin-S | LN | 49.61M | 8.75G | 140.9 | 436.9 |
| | BN+FFNBN | 49.68M | 8.76G | **181.6** | 376.6 |
| | BN+FFNBN* | 49.56M | 8.74G | - | **491.8** |
| Swin-B | LN | 87.77M | 15.44G | 91.21 | 278.1 |
| | BN+FFNBN | 87.86M | 15.45G | **120.3** | 262.2 |
| | BN+FFNBN* | 87.71M | 15.43G | - | **318.6** |

BN in each FFN block, leading to more FLOPs than LN. Second, normalization layers generally take a larger proportion of time cost in training. Thus BN+FFNBN saves more time than LN in training and therefore hide the extra time cost of BN in FFN. Also, with merging BN to linear layers, the inference speed could be further accelerated. Overall, BN-based Swin Transformers are about 20% faster than LN on average for both training and inference.

## 5. Conclusion

In this paper, we leverage the Batch Normalization to Transformer-based vision architectures. Our initial exploration reveals that the frequent crashes in model training when directly replacing all LN layers with BN is due to the un-normalized feed forward networks. We therefore propose to add a BN layer in-between the two linear layers in the FFN block where stabilized training statistics are observed, resulting in a pure BN-based architecture. Our experiments proved that our resulting approach is as effective as the LN-based counterpart and is about 20% faster in both training and inference due to the inherent advantages of BN. We hope that our exploration could motivate the future study on how to design better normalization strategies or architectures for vision Transformers.

## References


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

[2] Andrew Brock, Soham De, and Samuel L Smith. Characterizing signal propagation to close the performance gap in unnormalized resnets. *arXiv preprint arXiv:2101.08692*, 2021.

[3] Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised vision transformers. *arXiv preprint arXiv:2104.02057*, 2021.

[4] Z Dai and R Heckel. Channel normalization in convolutional neural networks avoids vanishing gradients. In *International Conference on Machine Learning, Workshop Deep Phenomena*, 2019.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[6] Xiaohan Ding, Yuchen Guo, Guiguang Ding, and Jungong Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1911–1920, 2019.

[7] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Diverse branch block: Building a convolution as an inception-like unit. *arXiv preprint arXiv:2103.13425*, 2021.

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[9] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[10] Boris Hanin and David Rolnick. How to start training: The effect of initialization and architecture. *arXiv preprint arXiv:1803.01719*, 2018.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[13] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[14] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.

[15] Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. *arXiv preprint arXiv:1702.03275*, 2017.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[19] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021.

[20] Chao Peng, Tete Xiao, Zeming Li, Yuning Jiang, Xiangyu Zhang, Kai Jia, Gang Yu, and Jian Sun. Megdet: A large mini-batch object detector. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6181–6189, 2018.

[21] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[22] Sheng Shen, Zhewei Yao, Amir Gholami, Michael Mahoney, and Kurt Keutzer. Rethinking batch normalization in transformers. *arXiv preprint arXiv:2003.07845*, 2020.

[23] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, May 2015.

[24] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[25] Masato Taki. Deep residual networks and weight initialization. *arXiv preprint arXiv:1709.02956*, 2017.

[26] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[27] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. *arXiv preprint arXiv:2012.12877*, 2020.

[28] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. *arXiv preprint arXiv:2103.17239*, 2021.

[29] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[31] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, et al. Deep high-resolution representation learning for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[32] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *arXiv preprint arXiv:2102.12122*, 2021.

[33] Bichen Wu, Chenfeng Xu, Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Zhicheng Yan, Masayoshi Tomizuka, Joseph Gonzalez, Kurt Keutzer, and Peter Vajda. Visual transformers: Token-based image representation and processing for computer vision. *arXiv preprint arXiv:2006.03677*, 2020.

[34] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. *arXiv preprint arXiv:2103.15808*, 2021.

[35] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[36] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pages 10524–10533. PMLR, 2020.

[37] Greg Yang and Samuel S Schoenholz. Mean field residual networks: On the edge of chaos. *arXiv preprint arXiv:1712.08969*, 2017.

[38] Zhuliang Yao, Yue Cao, Shuxin Zheng, Gao Huang, and Stephen Lin. Cross-iteration batch normalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12331–12340, June 2021.

[39] Kun Yuan, Shaopeng Guo, Ziwei Liu, Aojun Zhou, Fengwei Yu, and Wei Wu. Incorporating convolution designs into visual transformers. *arXiv preprint arXiv:2103.11816*, 2021.

[40] Daquan Zhou, Bingyi Kang, Xiaojie Jin, Linjie Yang, Xiaochen Lian, Zihang Jiang, Qibin Hou, and Jiashi Feng. Deepvit: Towards deeper vision transformer. *arXiv preprint arXiv:2103.11886*, 2021.