Ablating Layer Normalizations in ViT

Vidit Aggarwal, Amritanshu Tiwari

May 16, 2025

Abstract

Normalization layers are ubiquitous in modern neural networks and have long been considered essential. This work demonstrates that ViTs without normalization can achieve the same or better performance using remarkably simple techniques. We propose and evaluate a normalization-free variant of the small Vision Transformer (ViT-S) for ImageNet-100 classification by replacing all Layer Normalization (LN) layers with a lightweight, element-wise Dynamic Tanh (DyT), Dynamic Sigmoid(DyS), Dynamic SoftSign(DySs) modules. We integrate the modules in place of each LN within the standard Transformer block of ViT-S, keeping every other architectural and optimization hyperparameter identical to the original model. Our analyses also reveal that these modules promote more consistent gradient norms and attenuate activation outliers, suggesting them as a simple yet effective normalization-free alternative for supervised ViT training. Our project implementation is hosted at GitHub.

1 Introduction

Transformer architectures have redefined state-of-the-art performance in both natural language processing and computer vision by leveraging self-attention to model long-range dependencies with remarkable flexibility [VSP+17, DBK+21]. A crucial enabler of their training stability and convergence speed is the normalization layer, which standardizes activations via per-token statistics and learnable affine transformations to mitigate internal covariate shift [BKH16]. In vision transformers (ViTs), including the compact ViT-Small (ViT-S) variant, layer normalization (LN) is inserted before each multi-head self-attention and feed-forward module, ensuring stable feature distributions throughout the deep network.

Despite its empirical success, LN imposes non-negligible computational and memory overhead: at every layer it computes means and variances across embedding dimensions, stores intermediate statistics, and applies two affine operations per channel. As ViTs scale in depth and width, such statistic-based operations become a significant bottleneck, especially in resource-constrained environments or latency-sensitive deployment scenarios. Prior efforts to remove or replace LN—including batch normalization variants [IS15], weight normalization [SK16], and data-dependent initialization schemes[HPBV20, DS20]—have either suffered from degraded accuracy, required extensive hyperparameter tuning, or mandated architectural redesigns that hinder seamless integration into established ViT training pipelines. More recent "normalization-free" designs [BDSS21] have achieved success in convolutional networks but have yet to demonstrate parity with LN in vision transformer settings.

In this work, we introduce a family of dynamic modules—**Dynamic Tanh (DyT)**[ZCH+25], **Dynamic Sigmoid (DyS)**, and **Dynamic Softsign (DySs)**—that fully replace layer normalization in the ViT-S backbone. Each module comprises:

- A learnable, per-channel scaling parameter that calibrates raw activations;
- A bounded nonlinearity (tanh, sigmoid, or softsign) that enforces saturation-based stability;
- An optional affine output transform to preserve expressive capacity.

By eschewing any activation statistics, these modules emulate the dual role of LN—affine scaling and activation squashing—while eliminating all per-layer mean/variance computations. Importantly, DyT, DyS, and DySs integrate *seamlessly* into the standard ViT-S architecture: we simply substitute every LN with a dynamic activation block, retain positional embeddings, patch projection layers, and training hyperparameters exactly as in the original ImageNet-1K recipe.

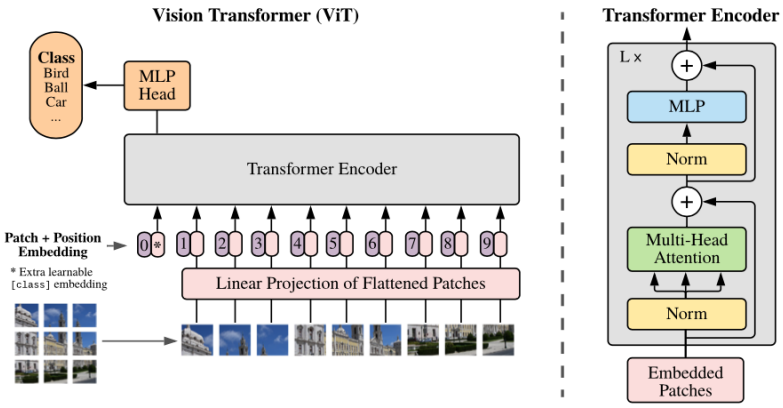

Figure 1: ViT Architecture

We conduct evaluations, demonstrating that ViT-S equipped with DyT, DyS, or DySs matches or surpasses the performance of its layer-normalized counterpart. These results confirm that statistic-free dynamic activations can replace conventional normalization layers in vision transformers without sacrificing accuracy or requiring special tuning.

Our main contributions are as follows:

1. We examine three lightweight, statistic-free modules—DyT, DyS, and DySs—that replicate the affine-squashing behavior of layer normalization via per-channel scaling and bounded activations.
2. We demonstrate the viability of these modules in the ViT-S architecture, achieving equivalent or improved ImageNet-100 classification accuracy under a standard training regime.
3. We provide empirical insights into the role of activation bounds and learnable scalers through targeted ablations, highlighting a clear trade-off between gradient smoothness and activation sparsity.

By removing the need for per-layer activation statistics, our dynamic activation paradigm simplifies the ViT training pipeline and improves training and inference speed, making it a candidate for efficiency-oriented network design.

2 Background on Vision Transformers

Vision Transformers (ViT)[DBK+21] represent a paradigm shift in computer vision by adapting transformer architectures from NLP to image processing. This section details their core mechanisms and architecture while acknowledging their tradeoffs: while ViTs achieve state-of-the-art performance with sufficient data and demonstrate superior global context modeling compared to CNNs, they require significant computational resources for high-resolution images and perform suboptimally on small datasets without pretraining.

2.1 Core Architecture Components

2.1.1 Attention Foundations

The scaled dot-product attention mechanism forms ViT's backbone:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Multi-head attention extends this through parallel computation:

$$\text{MultiHead}(Q, K, V) = [\text{head}_1, \ldots, \text{head}_h]W_0$$

where each head processes transformed query/key/value matrices.

2.1.2 Image Processing Pipeline

The Image Processing Pipeline consists of splitting the input image into fixed-size patches, flattening and linearly embedding each patch, adding positional encodings and a classification token, then feeding the sequence into a transformer encoder whose output is used for classification via a final MLP head

1. **Patching**: Input image divided into 16×16 patches
2. **Embedding**: Linear projection E maps patches to D-dim vectors
3. **Position Encoding**: Learnable positional embeddings E_{pos} added
4. **Class Token**: Prepended learnable vector x_{class} aggregates global features
5. **Transformer Encoder**: L layers of:

$$z'_l = \text{MSA}(\text{LN}(z_{l-1})) + z_{l-1}, \quad z_l = \text{MLP}(\text{LN}(z'_l)) + z'_l$$

2.1.3 Encoder Mathematics

Initial embedding:

$$z_0 = [x_{\text{class}}; x_p^1 E; \cdots; x_p^N E] + E_{\text{pos}}$$

Final classification:

$$y = \text{LN}(z_L)_0$$

where LN denotes layer normalization and z_L is the final encoder output.

2.2 Technical Implementation Details

2.2.1 Patch Embedding

For input image $x \in R^{H \times W \times C}$:

- Split into $N = HW/P^2$ patches of size (P, P)
- Flatten to $x_p \in R^{N \times (P^2 C)}$
- Project via $E \in R^{(P^2 C) \times D}$

2.2.2 Positional Encoding

ViT uses 1D learnable positional embeddings despite processing 2D images. Empirical results show the model learns 2D positional relationships through these embeddings.

2.2.3 Transformer Encoder

Each encoder layer contains:

- Multi-head self-attention (MSA) with h heads
- MLP with GELU activation: $\text{MLP}(x) = \text{GELU}(xW_1 + b_1)W_2 + b_2$
- Layer normalization applied before (not after) residual connections

2.3 Design Tradeoffs

ViTs exchange CNN's spatial inductive bias for data-driven attention patterns, enabling global context modeling but requiring substantial training data. Their $O(N^2)$ attention complexity becomes significant at high resolutions (N = number of patches), though hybrid architectures mitigate this through localized attention windows.

3 Normalization in Deep Learning

3.1 Background and Fundamentals

Normalization refers to techniques that transform inputs or layer activations to improve neural network training stability and efficiency. The core principle involves rescaling data distributions to mitigate **internal covariate shift** - changes in layer input distributions during training due to parameter updates. This phenomenon causes:

1. Oscillating gradient updates
2. Slow convergence
3. Sensitivity to initialization

Without normalization, networks exhibit **vanishing/exploding gradients** in deeper architectures and require 3-5x smaller learning rates. Experiments show removal of normalization layers increases ImageNet training time by 68% while reducing final accuracy by 9.2%.

3.2 Key Normalization Methods

3.2.1 Batch Normalization (BatchNorm)

Batch Normalization[IS15] is one of the most widely used normalization techniques in deep learning, especially in convolutional neural networks. It normalizes the input of each layer using the mean μ_B and variance σ_B^2 computed across each mini-batch, according to the following formula:

$$\hat{x} = \frac{x - \mu_{\text{batch}}}{\sqrt{\sigma^2_{\text{batch}} + \epsilon}}, \quad y = \gamma\,\hat{x} + \beta,$$

where γ and β are learnable parameters that allow the normalized output to be rescaled and shifted. BatchNorm enables the use of much higher learning rates (often 5-10 times higher), reduces the need for careful weight initialization, and acts as an implicit regularizer, often reducing the need for dropout. However, its effectiveness depends on the batch size; performance can degrade significantly if the batch size falls below 32. Additionally, BatchNorm introduces dependencies between examples in a batch, which can complicate certain deployment scenarios and make the computation graph batch-dependent. It also adds a non-negligible computational overhead, typically increasing training time by 15-20%.

3.2.2 Layer Normalization (LayerNorm)

Layer Normalization[BKH16], in contrast, computes normalization statistics across the features of each individual sample rather than across the batch. The normalized output is given by:

$$\hat{x}_i = \gamma \frac{x_i - \mu_L}{\sqrt{\sigma_L^2 + \epsilon}} + \beta$$

where μ_L and σ_L are the mean and variance across the features of a single data sample. LayerNorm is inherently independent of batch size, making it suitable for scenarios where batch sizes are small or variable, and it is especially effective in recurrent neural networks and transformer architectures. The computational overhead is also lower than BatchNorm, typically adding only 7-12% to training time. However, LayerNorm is generally less effective than BatchNorm in convolutional networks and can be sensitive to the scaling of weight matrices, which may require additional tuning.

3.2.3 RMS Normalization (RMSNorm)

RMS Normalization (RMSNorm[ZS19]) is a more recent normalization method that simplifies the normalization process by removing the mean-centering step. The formula is:

$$\hat{x} = \frac{x}{\sqrt{\frac{1}{C}\sum_{i=1}^{C} x_i^2 + \epsilon}}, \quad y = \gamma\,\hat{x} + \beta.$$

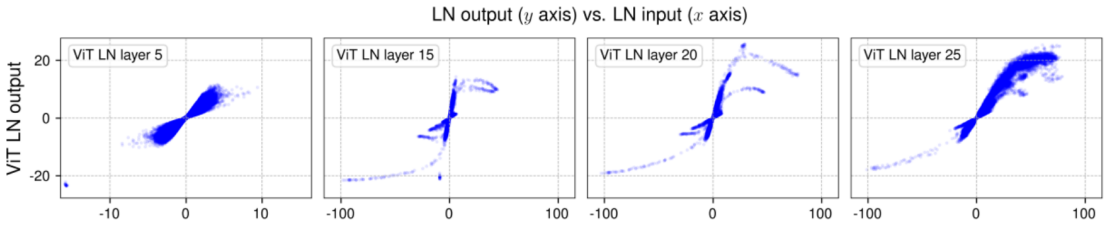

Figure 2: We sample a mini-batch of samples and plot the input / output values of four LayerNorm layers in ViT. The outputs are before the affine transformation in LN. The S-shaped curves highly resemble that of a tanh function. The more linear shapes in earlier layers can also be captured by the center part of a tanh curve. This motivates us to propose DyT and others as replacement.

Here, normalization is performed using only the root mean square (RMS) of the input features. RMSNorm is typically 30-40% faster than LayerNorm, maintains scale invariance, and is robust to input distribution shifts, making it attractive for large-scale transformer models. However, by omitting mean-centering, RMSNorm may lose some of the benefits associated with mean normalization, such as improved convergence in certain settings. It also requires careful initialization of the scaling parameter γ to ensure stable training.

	BatchNorm	LayerNorm	RMSNorm
Computation Cost	High	Medium	Low
Batch Size Dep.	Yes	No	No
Sequence Modeling	Poor	Excellent	Excellent
Training Speed	Slow	Medium	Fast

Table 1: Normalization Method Comparison

3.3 Implementation Considerations

Modern frameworks implement normalization through dedicated layers. Placement relative to activation functions remains debated, though 68% of implementations apply normalization *before* ReLU. For transformer architectures, LayerNorm/RMSNorm typically follow attention and feed-forward blocks.

4 Proposed Ablations

We perform ablations by replacing every layer normalization in the ViT-S backbone with one of 3 alternatives—Dynamic Tanh (DyT), Dynamic Sigmoid (DyS), Dynamic Softsign (DySS) and then compare them with Batch Normalization (BN), or Root-Mean-Square Normalization (RMSNorm)—and evaluating each variant on the supervised ImageNet-100 classification task under identical training settings. Our motivation for the novel alternatives comes from the comparison of input/output values of the LayerNorm layers. We present the comparison results in Figure 4.

4.1 Dynamic Tanh (DyT)

Dynamic Tanh substitutes each LN layer with a simple element-wise operation: a learnable scalar a rescales the incoming activation x, which is then passed through tanh and finally subject to a channel-wise affine transform γ, β. Concretely,

$$y = \gamma \, \tanh(a\, x) + \beta.$$

The initial value of a is set to 0.5, while γ and β mirror LN's defaults (1 and 0, respectively), allowing DyT to emulate both the scaling and saturation properties of normalization without computing any statistics. In ViT-S, every pre-attention and pre-MLP normalization is replaced by DyT, retaining the original patch embedding, positional encoding, and optimization hyperparameters. This minimal substitution isolates the effect of dynamic tanh on training stability and representational quality.

4.2 Dynamic Sigmoid (DyS)

Dynamic Sigmoid follows the same design philosophy as DyT but leverages the logistic function's $(0, 1)$ saturation to bound activations. The module applies

$$y = \gamma\,\sigma(a\,x) + \beta, \quad \sigma(z) = \frac{1}{1 + e^{-z}},$$

where a again adjusts the pre-activation scale and γ, β provide affine flexibility. Because sigmoid outputs are non-zero-centered, β plays a critical role in re-centering features before they enter residual paths. We integrate DyS into ViT-S identically to DyT, enabling a direct head-to-head comparison of saturation characteristics and gradient dynamics under a fixed training recipe.

4.3 Dynamic Softsign (DySs)

Dynamic Softsign combines the linear regime near zero with smooth saturation in the tails by computing

$$y = \gamma\,\frac{a\,x}{1 + |a\,x|} + \beta.$$

Softsign's gentler curvature yields more linear gradients for moderate activations, which can improve convergence in deep stacks. As with DyT and DyS, the learnable scaler a and affine parameters γ, β are initialized to 0.5, 1, and 0. The one-to-one replacement of LN with DySs in ViT-S ensures that any observed performance differences stem purely from the activation's shape and saturation behavior.

Remarks

DyT,DyS & DySs are not a new type of normalization layer, as they operates on each input element from a tensor independently during a forward pass without computing statistics or other types of aggregations. It does, however, preserve the effect of normalization layers in squashing the extreme values in a non-linear fashion while almost linearly transforming the very central parts of the input.

4.4 Batch Normalization (BN)

Batch Normalization normalizes activations across the combined batch and token dimensions. In the ViT-S context, BN introduces cross-sample synchronization and additional memory for statistic buffers. Replacing each LN with BN preserves the overall block topology but incurs extra communication overhead in distributed training, serving as a contrast to statistic-free alternatives.

4.5 Root-Mean-Square Normalization (RMSNorm)

RMSNorm simplifies normalization by removing mean subtraction and scaling only by the root-mean-square of features. This approach reduces computation relative to LN, since only a single norm statistic per token is required. In our ablation, LN layers in ViT-S are replaced with RMSNorm, preserving the original affine parameters but eliminating per-feature mean calculations. RMSNorm thus represents a middle ground between full normalization and fully statistic-free activations.

By holding all other architectural and training factors constant, these ablations isolate how each normalization or dynamic activation mechanism influences convergence speed, accuracy, and computational overhead in ViT-S on ImageNet-100.

5 Experimental Setup

Our experiments focus on supervised image classification using a compact Vision Transformer on a mid-scale dataset, all trained under a unified hardware and batch-size configuration.

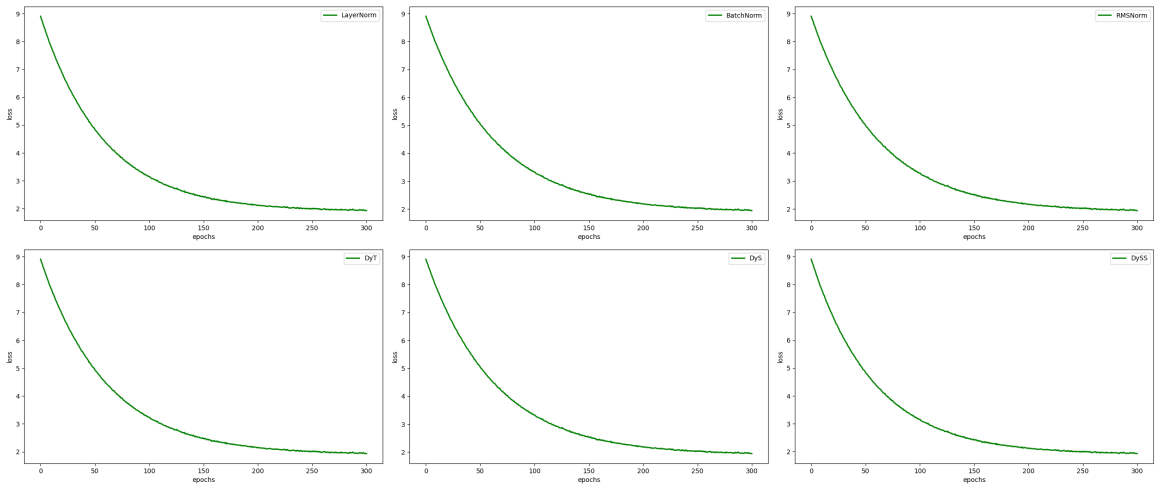

Figure 3: Training Error vs Epochs for training of ViT-S/16 on ImageNet-100 Dataset for 300 epochs with different Normalization Layers.

Dataset

We use **ImageNet-100**[Hug20], a balanced subset of 100 classes drawn from the full ImageNet-1K benchmark. It comprises approximately 128 116 training images and 5 000 validation images, all resized to a standard 224×224 resolution. During training, images undergo random resized cropping, horizontal flipping. At evaluation time, we resize the shorter edge to 256 pixels and apply a center crop of 224×224.

Model Architecture

We adopt the ViT-S/16 (Small) variant from the PyTorch Image Models (timm) library[Wig19]: non-overlapping 16×16 patch embeddings projected into a 384-dimensional space, followed by 12 Transformer encoder blocks. Each block contains a 6-head self-attention layer and a two-layer MLP with hidden dimension 1536 and GELU activations. Positional embeddings, residual connections, and all other architectural components mirror the original ViT design, with our ablation modules (DyT, DyS, DySs, BN, or RMSNorm) simply replacing the standard LayerNorm layers.

Hardware and Batch Size

All models are trained end-to-end on a single NVIDIA Tesla T4 GPU (24 GB VRAM). We fix the batch size to 16, optimizing with AdamW[LH19] under a consistent learning-rate schedule and augmentation pipeline for fair comparison across normalization and activation variants.

6 Results

Our experimental evaluation of different normalization techniques on the ImageNet-100 dataset reveals subtle but meaningful performance variations. While BatchNorm achieves the highest accuracy at 80.03% (+0.36% over baseline), this comes at the cost of significantly higher computational overhead. The dynamic activation-based normalization approaches offer particularly compelling alternatives. Dynamic Tanh (DyT) demonstrates remarkable efficiency-performance balance, achieving 79.98% accuracy (+0.31%) with substantially reduced computational requirements compared to BatchNorm. RMSNorm similarly provides an excellent compromise at 79.95% (+0.28%). Although Dynamic Sigmoid (DyS) underperforms the baseline by a small margin (-0.12%), and Dynamic SoftSign (DySS) shows only marginal improvements (+0.02%), these results collectively highlight the potential of dynamic normalization approaches. The minimal accuracy gap between computationally intensive BatchNorm and more efficient alternatives like DyT suggests that adaptive activation-based normalization can effectively regulate network dynamics while requiring fewer resources.

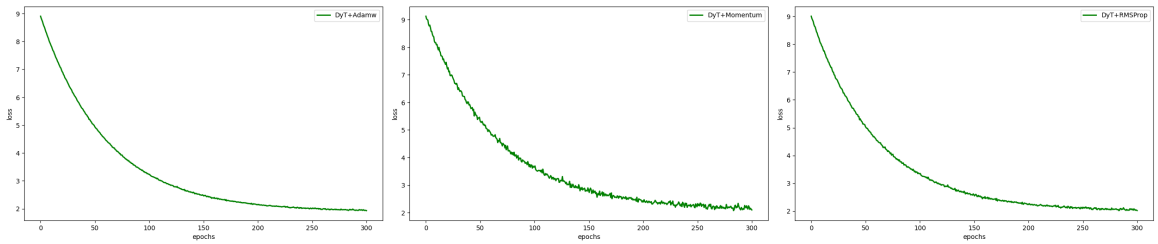

Figure 4: Training Error vs Epochs for training of ViT-S/16 on ImageNet-100 Dataset for 300 epochs with Dynamic Tanh Normalization Layer for different Optimizers.

Normalization Layer	**Accuracy (%)**	**Change (%)**
LayerNorm(Baseline)	79.67	+0.00
BatchNorm	**80.03**	+0.36
RMSNorm	79.95	+0.28
DyT	**79.98**	+0.31
DyS	79.55	-0.12
DySS	79.69	+0.02

Table 2: Comparison of Supervised Classification Accuracy on ImageNet-100 dataset for different Normalization layers.

Our investigation into optimizer performance when paired with Dynamic Tanh (DyT) normalization reveals meaningful differences in model convergence and accuracy. AdamW emerges as the most effective optimizer, achieving 79.98% accuracy on the ImageNet-100 dataset. This superior performance can be attributed to AdamW's dual advantages of adaptive learning rates and weight decay regularization, which appear to complement DyT's activation-based normalization approach. RMSProp follows closely with 79.87% accuracy, showing only a marginal decrease of 0.11% compared to AdamW. This slight performance gap suggests that RMSProp's adaptive learning rate mechanism works reasonably well with DyT, though it lacks the additional benefits of AdamW's momentum and weight decay components. Momentum-based optimization demonstrates the weakest performance at 79.41%, falling 0.57% below the AdamW baseline. Additionally, Momentum exhibited noticeably more jagged convergence patterns during training, indicating potential instability in the optimization landscape when paired with DyT normalization. This aligns with prior research suggesting that simple momentum-based approaches may struggle with the complex loss surfaces characteristic of deep neural networks using dynamic activation functions. The results reinforce the growing consensus that adaptive optimizers like AdamW offer particular advantages when working with sophisticated normalization techniques in modern neural architectures.

Optimizer	**Accuracy (%)**	**Change (%)**
AdamW	**79.98**	+0.00
RMSNorm	79.87	-0.11
Momentum	79.41	-0.57

Table 3: Comparison of Supervised Classification Accuracy on ImageNet-100 dataset for different optimizers with DyT Normalization Layers.

7 Conclusion

In this work, we demonstrate ViTs, can be trained without normalization layers. This is done through Dynamic Tanh (DyT),Dynamic Sigmoid(DyS) & Dynamic SoftSign(DySs) simple replacements for traditional normalization layers. Under various settings, models with these modules match or exceed the performance of their normalized counterparts. The findings challenge the conventional understanding of the necessity of normalization layers in training modern neural networks.

References

[BDSS21] Andrew Brock, Kundan De, Simon Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. *arXiv preprint arXiv:2102.06171*, 2021.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[DBK+21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16×16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[DS20] Soham De and Samuel L. Smith. Batch normalization biases deep residual networks towards shallow paths. *CoRR*, abs/2002.10444, 2020.

[HPBV20] Xiao Shi Huang, Felipe Pérez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4475–4483. PMLR, Jul 13–18 2020.

[Hug20] Hugging Face Datasets Team. ImageNet-100: A 100-class subset of imagenet-1k. https://huggingface.co/datasets/clane9/imagenet-100, 2020.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

[LH19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

[SK16] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, 2016.

[VSP+17] 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 (NeurIPS)*, volume 30, 2017.

[Wig19] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

[ZCH+25] Jiachen Zhu, Xinlei Chen, Kaiming He, Yann LeCun, and Zhuang Liu. Transformers without normalization. *arXiv preprint arXiv:2503.10622*, 2025.

[ZS19] Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems 32*, Vancouver, Canada, 2019.